Where are we now? A large benchmark study of recent symbolic regression methods

PATRYK ORZECHOWSKI, University of Pennsylvania
WILLIAM LA CAVA*, University of Pennsylvania
JASON H. MOORE, University of Pennsylvania

In this paper we provide a broad benchmarking of recent genetic programming approaches to symbolic regression in the context of state of the art machine learning approaches. We use a set of nearly 100 regression benchmark problems culled from open source repositories across the web. We conduct a rigorous benchmarking of four recent symbolic regression approaches as well as nine machine learning approaches from scikit-learn. The results suggest that symbolic regression performs strongly compared to state-of-the-art gradient boosting algorithms, although in terms of running times is among the slowest of the available methodologies. We discuss the results in detail and point to future research directions that may allow symbolic regression to gain wider adoption in the machine learning community.

CCS Concepts: • **Computing methodologies → Classification and regression trees**; **Genetic programming**; Ensemble methods; Cross-validation;

Additional Key Words and Phrases: symbolic regression, benchmarking, machine learning, genetic programming

ACM Reference Format:
Patryk Orzechowski, William La Cava, and Jason H. Moore. 2018. Where are we now? A large benchmark study of recent symbolic regression methods. 1, 1 (June 2018), 12 pages. https://doi.org/10.1145/3205455.3205539

1 INTRODUCTION

Since the beginning of the field, the genetic programming (GP) community has considered the task of symbolic regression (SR) as a basis for methodology research and as a primary application area. GP-based SR (GPSR) has produced a number of notable results in real-world regression applications, for example dynamical system modeling in physics [28], biology [30], industrial wind turbines [19], fluid dynamics [18], robotics [2], climate change forecasting [33], and financial trading [17], among others. However, the most prevalent use of GPSR is in the experimental analysis of new methods, for which SR provides a convenient platform for benchmarking. Despite this persistent use, several shortcomings of SR benchmarking are notable. First, the GP community lacks a unified standard for SR benchmark *datasets*, as noted previously [21]. Several SR benchmarks have been proposed [17, 23, 36], critiqued [6, 21], and black-listed [21], leading to inconsistencies in the experimental design of papers. In addition to a lack of consensus for

*corresponding author

Authors' addresses: Patryk Orzechowski, University of Pennsylvania, 3700 Hamilton Walk, Philadelphia, PA 19104, USA, patryk.orzechowski@gmail.com; William La Cava, University of Pennsylvania, 3700 Hamilton Walk, Philadelphia, PA 19104, USA, lacava@upenn.edu; Jason H. Moore, University of Pennsylvania, 3700 Hamilton Walk, Philadelphia, PA 19104, jhmoore@upenn.edu.

benchmark datasets, there is not a standard set of benchmark *algorithms* against which new methods are compared. As a result, it is typical for researchers to design or choose their own set of algorithms to compare to proposed methods, and it is up to reviewers and readers to assess the validity of the comparison. Experiments typically consider single values for GP hyperparameters such as population size or crossover rate, which increases the uncertainty of results even further. These practices make it nearly impossible to judge a new method outside the narrow scope of the experimental results.

Of course, there are shortcomings to focusing on benchmarks as well, as noted by others [9, 38]. Putting too much focus on benchmarking may stifle innovation or lead to a lack of generalization to new tasks. However, the evidence suggests that the GP community is far from being overly focused on benchmarking. A 2012 survey of GP papers in EuroGP and GECCO from 2009 - 2011 reported the average number of SR problems per paper to be 2.4 [21]; 26.2% of papers relied on the quartic polynomial problem, which has since been black-listed for being too trivial [38]. We contend that the lack of focus in the GP community on rigorous benchmarking makes it hard to know how GPSR methods fit into the broader machine learning (ML) community. This lack of clarity also impedes the adoption of advancements to traditional GP techniques, and leaves researchers unsure about which advancements will have meaningful impacts.

There have been a few efforts to conduct broad benchmarking of GP methods in the past. For example, a recent a study looked at five SR methods on a set of five synthetic and four real world datasets [37]. Outside of GP, the efforts to benchmark ML approaches across many problems are more frequent, although most focus on the task of classification. Previous studies have looked at hundreds classification methodologies [11] and up to 165 datasets [24]. Collaborative online tools such as Kaggle and OpenML [35] have also driven ML benchmarking and adoption of new methods. These larger benchmark studies have, for the most part, ignored GP-based methods. As a result, the GPSR field lacks a general sense of where it stands in relation to the broader ML field in terms of expected performance.

Our goal in this study is to present initial results in our efforts to assess the performance of recent GPSR methods in the broad context of ML regression. We benchmark the performance of four recent SR algorithms and ten established ML approaches on a collection of 94 different real-world regression problems. For each problem we consider hyperparameter tuning via cross-validation and assess each method in terms of training error, test error, and wall-clock time. Finally, we provide the code for the analysis in order to allow researchers to benchmark their own methods in this framework and reproduce the results shown here.

2 METHODS

We compare four recent GPSR methods in this benchmark and ten well-established ML regression methods. In this section we briefly present the selected methods and describe the design of the experiment.

2.1 GP methods

A number of factors impacted our choice of these methods. Two key elements were open-source implementations and ease of use. In addition, we wished to test different research thrusts in GP literature. The four methods encompass different innovations to standard GPSR, including incorporation of constant optimization, semantic search divers, and Pareto optimization. Each method is described briefly below.

Multiple regression genetic programming (MRGP). [1] MRGP combines Lasso regression with the tree search afforded by GP. A weight is attached to each node in each program. These weights are adapted by applying Lasso regression to the entire program trace. MRGP uses point mutation and sub-tree crossover for variation and NSGA-II for selection. We use the version implemented in FlexGP [1].

ϵ-Lexicase selection (EPLEX). [20] ϵ-lexicase selection adapts lexicase selection method [32] for regression. Rather than aggregating performance on the training set into a single fitness score, EPLEX selects parents by filtering the population through randomized orderings of training samples and removing individuals that are not within ϵ of the best performance in the pool. We use the EPLEX method implemented in ellyn[2]. Ellyn is a stack-based GP system written in C++ with a Python interface for use with scikit-learn. It uses point mutation and subtree crossover. Weights in the programs are trained each generation via stochastic hill climbing. A Pareto archive of trade-offs between mean squared error and complexity is kept during each run, and a small internal validation fold is used to select the final model returned by the search process.

Age-fitness Pareto Optimization (AFP). [29] AFP is a selection scheme based on the concept of age-layered populations introduced by Hornby et. al. [15]. AFP introduces a new individual each generation with an age of 0. An individual's age is updated each generation to reflect the number of generations since its oldest node (gene) entered the population. Parent selection is random and Pareto tournaments are used for survival on the basis of age and fitness. We use the version of AFP implemented in ellyn, with the same settings described above.

Geometric Semantic Genetic Programming (GSGP). [22] GSGP is a recent method that has shown many promising results for SR and other tasks. The main concept behind GSGP is the use of semantic variation operators that produce offspring whose semantics lie on the vector between the semantics of the parent and the target semantics (i.e. target labels). Use of these variation operators has the advantage of creating a unimodal fitness landscape. On the downside, the variation operators result in exponential growth of programs. We use the version GSGP implemented in C++ by Castelli et. al. [4], which is optimized to minimize memory usage. It is available from SourceForge[3].

2.2 ML methods

We use scikit-learn [26] implementations of the following methods in this study:

Linear Regression. Linear Regression is a simple model of regression that minimizes the sum of the square errors of a linear model of inputs. The model is defined by $\hat{y} = b + w^T x$, where y is a dependent variable (target), x are explanatory variables, b and w are intercept and slope variables, and the minimized function is equal to (1).

$$C_{LR}(w) = \frac{1}{2}\sum_i (y_i - w^T x_i)^2 \tag{1}$$

Kernel Ridge. Kernel Ridge [27] performs Ridge regression using a linear function in the space of the respective kernel. Least squares with l2-norm regularization is applied in order to prevent overfitting. The minimized function is equal to (2), where ϕ is a kernel function and λ is the regularization parameter.

$$C_{KR}(w) = \frac{1}{2}\sum_i (y_i - w^T \phi(x_i))^2 + \frac{1}{2}\lambda ||w||^2 \tag{2}$$

Least-angle regression with Lasso. Lasso (Least absolute shrinkage and selection operator) is a popular method of regression that applies both feature selection and regularization [34]. Similarly to Kernel Ridge, high values of w are

[1]https://flexgp.github.io/gp-learners/
[2]https://epistasislab.github.io/ellyn/
[3]http://gsgp.sourceforge.net/

penalized. The use of the l1-norm on w in the minimization function (see (3)) improves the ability to push individual weights to zero, effectively performing feature selection.

$$C_L(w) = \frac{1}{2}\sum_i (y_i - w^T \phi(x_i))^2 + \lambda ||w||_1 \tag{3}$$

Least-angle regression with Lasso, a.k.a. Lars [10], is an efficient algorithm for producing a family of Lasso solutions. It is able to compute the exact values of λ for new variables entering the model.

Linear SVR. Linear Support Vector Regression extends the concept of Support Vector Classifiers (SVC) to the task of regression, i.e. to predict real values instead of classes. Its objective is to minimize an ϵ-insensitive loss function with a regularization penalty ($\frac{1}{2}||w||^2$) in order to improve generalization [31].

SGD Regression. SGD Regression implements stochastic gradient descent and is especially well suited for larger problems with over 10,000 of instances [26]. We add this method of regression regardless, to compare its performance on smaller datasets.

MLP Regressor. Neural networks have been applied to regression problems for almost three decades [14]. We include multilayer perceptrons (MLPs) as one of the benchmarked algorithms. We decided to benchmark neural network with a single hidden layer with fixed number of neurons (100) and compare different activation functions, learning functions and solvers, including the novel adam solver [16].

AdaBoost regression. Adaptive Boosting, called also AdaBoost [8, 12], is a flexible technique of combining a set of weak learners into a single stronger regressor. By changing the distribution (i.e. weights) of instances in the data, previously misclassified instances are favored in consecutive iterations. The final prediction is obtained by a weighted sum or weighted majority voting. As the result, the final regressor has smaller prediction errors. The method is considered sensitive to outliers.

Random Forest regression. Random Forests [3] are a very popular ensemble method based on combining multiple decision trees into a single stronger predictor. Each tree is trained independently with a randomly selected subset of the instances, in a process known as bootstrap-aggregating or bagging. The resulting prediction is an average of multiple predictions. Random forests try to reduce variance by not allowing decision trees to grow large, making them harder to overfit.

Gradient Boosting regression. Gradient Boosting [13] is an ensemble method that is based on regression trees. It shares the AdaBoost concept of iteratively improving the system performance on its weakest points. In contrast to AdaBoost, the distribution of the samples remain the same. Instead, consecutively created trees correct the errors of the previous ones. Gradient Boosting minimizes bias (not variance like in Random Forests). In comparison to Random Forests, Gradient Boosting is sequential (thus slower), more difficult to train, but is reported to perform better than Random Forest [24].

Extreme Gradient Boosting. Extreme Gradient Boosting, also known as XGBoost [5], incorporates regularization into the Gradient Boosting algorithm in order to control overfitting. Its objective function combines the optimization of training loss with model complexity. This brings the predictor closer to the underlying distribution of the data, while encouraging simple models, which have smaller variance. Extreme gradient boosting is considered a state-of-the-art method in ML.

2.3 Datasets

We pulled the benchmark datasets from the Penn Machine Learning Benchmark (PMLB) [25] repository[4], which contains a large collection of standardized datasets for classification and regression problems. This repository overlaps heavily with datasets from UCI, OpenML, and Kaggle. In this paper we considered regression problems only, of which there are 120 total. For our experiment, we removed datasets with 3000 instances or more (22 datasets) and two others for which at least one of the methods failed to provide the required number of results in feasible time (i.e. 72 hours on Intel® Xeon® CPU E5-2680 v3 @ 2.50GHz). This gave the collection of 94 datasets in total. The distribution of the number of instances and the number of features in the collection of the datasets is presented in Fig. 1.

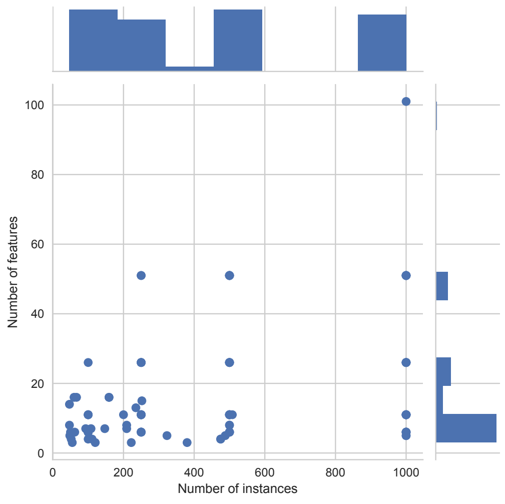

Fig. 1. Basic characteristics of the datasets used in the study

2.4 Experiment design

In order to benchmark different regression methods, an effort was made to measure performance of each of the methods in as similar an environment as possible. First, we decided to treat each of the GP methods as a classical ML approach and used the scikit-learn library [26] for cross validation and hyperparameter optimization. This required some source code modifications to allow GSGP and MRGP to communicate with the wrapper. Second, instead of reimplementing the algorithms, we relied on the original implementations with as few modifications as possible. Wrapping each method allowed us to keep a common benchmarking framework based on the scikit-learn functions.

The datasets were divided in the following way: 75% of samples in each of the datasets were used for training, whereas the remaining 25% were used for testing. We used grid search to tune the hyperparameters of each method. Each method was run with a preset grid of input parameters, detailed in Table 1. The optimal setting of the parameters was determined based on 5-fold cross-validation performed on training data only. The setting with the best R^2 score across all folds was used for training the algorithms on the whole training data. The performance of the methods was

[4]https://github.com/EpistasisLab/penn-ml-benchmarks

Table 1. Analyzed algorithms with their parameters settings. The parameters in quotations refer to their names in the scikit-learn implementations.

Algorithm name	Parameter	Values
eplex,	pop size / generations	{100/1000,1000/100}
afp,	max program length / max depth	{64 / 6}
mrgp	crossover rate	{0.2,0.5,0.8}
	mutation rate	1-crossover rate
gsgp	pop size / generations	{100/1000,200/500,1000/100}
	initial depth	{6}
	crossover rate	{0.0,0.1,0.2}
	mutation rate	1-crossover rate
eplex_1M	pop size / generations	{500/2000,1000/1000,2000/500}
	max program length	{100}
	crossover rate	{0.2,0.5,0.8}
	mutation rate	1-crossover rate
AdaBoostRegressor	‘n_estimators’	{10, 100, 1000}
	‘learning_rate’	{0.01, 0.1, 1, 10}
GradientBoostingRegressor	‘n_estimators’	{10, 100, 1000}
	‘min_weight_fraction_leaf’	{0.0, 0.25, 0.5}
	‘max_features’	{‘sqrt’,‘log2’, None}
KernelRidge	‘kernel’	{‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’}
	‘alpha’	{1e-4, 1e-2, 0.1, 1}
	‘gamma’	{0.01, 0.1, 1, 10 }
LassoLARS	‘alpha’	{ 1e-04, 0.001, 0.01, 0.1, 1 }
LinearRegression	default	default
MLPRegressor	‘activation’	{‘logistic’, ‘tanh’, ‘relu’}
	‘solver’	{‘lbfgs’,’adam’,‘sgd’}
	‘learning_rate’	{‘constant’, ‘invscaling’, ‘adaptive’}
RandomForestRegressor	‘n_estimators’	{10, 100, 1000}
	‘min_weight_fraction_leaf’	{0.0, 0.25, 0.5}
	‘max_features’	{‘’sqrt’,‘log2’, None}
SGDRegressor	‘alpha’	{1e-06, 1e-04, 0.01, 1 }
	‘penalty’	{‘l2’, ‘l1’, ‘elasticnet’}
LinearSVR	‘C’	{1e-06, 1e-04, 0.1, 1 }
	‘loss’	{‘epsilon_insensitive’, ‘squared_epsilon_insensitive’}
XGBoost	‘n_estimators’	{10, 50, 100, 250, 500, 1000}
	‘learning_rate’	{1e-4, 0.01, 0.05, 0.1, 0.2}
	‘gamma’	{0, 0.1, 0.2, 0.3, 0.4}
	‘max_depth’	{6}
	‘subsample’	{0.5, 0.75, 1}

measured on both training and testing datasets on the best model obtained during cross-validation. We repeated the entire experiment 10 times for each method and dataset.

Because of time constraints, we decided to run each of the GP-based methods for 100,000 evaluations (population size x number of generations). Additionally, we generated results for 1 million evaluations using EPLEX (referred to as EPLEX_1M) in order to assess how much a more thorough training of a GP-based regressor would improve its performance.

Data preprocessing. We decided to feed benchmarked algorithms with scaled data using StandardScaler function from scikit-learn. The reason for this is our effort to keep the format of the input data consistent across multiple algorithms for the purpose of benchmarking. The choice of the optimal preprocessing method for the particular regressor is out of scope of this paper.

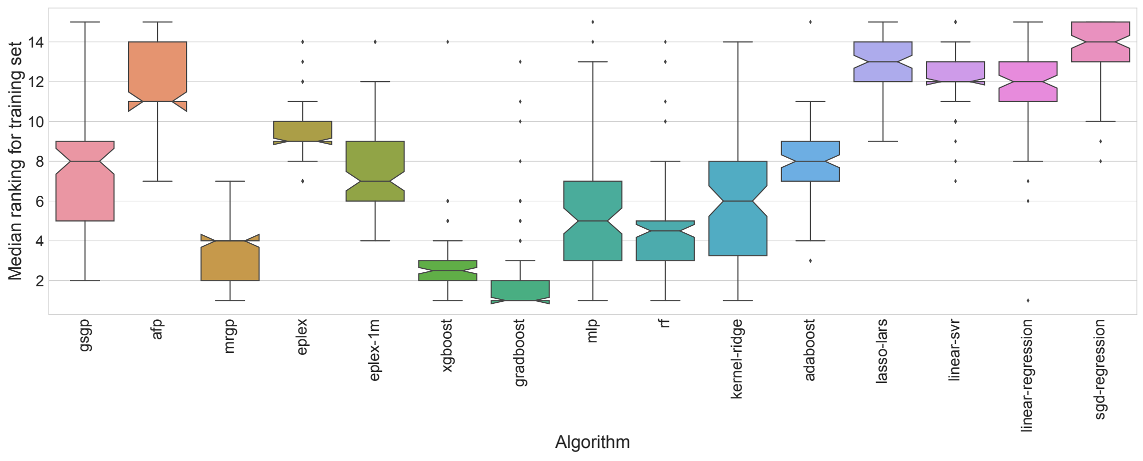

Fig. 2. Ranking of the performance of the algorithms based on the MSE score on training datasets.

Initialization of the algorithms. We initially considered starting each of the methods with the same random seed, but eventually decided to make all data splits randomly. In our belief both approaches have disadvantages: the results will either be biased by the choice of the random seed, or by using different splits for different methods. By taking a median of the scores we became independent of the initial split of the data.

Wrappers for the GP methods. Some modifications needed to be done to each of the GP methods. For EPLEX and AFP, ellyn provides an existing Python wrapper that was used. For other methods we implemented a class derived from scikit-learn *BaseEstimator*, which implemented two methods: *fit()*, used for training the regressor, and *predict()*, used for testing performance of the regressor. The source code of MRGP and GSGP had to be modified, so that the algorithms could communicate with the wrapper.

Parameters for the algorithms. The settings of the input parameters for the algorithms were determined based on the available recommendations for the given method, as well as previous experience of the authors. For GP-based methods we applied from 6 to 9 different settings (mainly: population size x number of generations and crossover and mutation rates). For the ML algorithms the number of settings was method dependent. The largest grid of the parameters was used for XGBoost method. The exact parameters for the methods used in this study can be found in Table 1.

3 RESULTS

We present aggregated results of the benchmarked algorithms on the collection of 94 regression datasets in Figures 2-3. The relative performance of the algorithms was determined as the ability to make the best predictions on the training and testing data using mean squared error (MSE) of the samples. The performance on the testing dataset is of primary importance, as it shows how well the methods can generalize to previously unseen data [7]. However we include the training comparisons as a way to assess the prediliction for overfitting among methods.

We first analyze the results for each of the regression tasks on the training data. The relative rankings of each method in terms of MSE is presented in Fig. 2. The best training performance was obtained with gradient boosting, which completed in top-2 for the vast majority of the benchmarked datasets. The second best method across all the

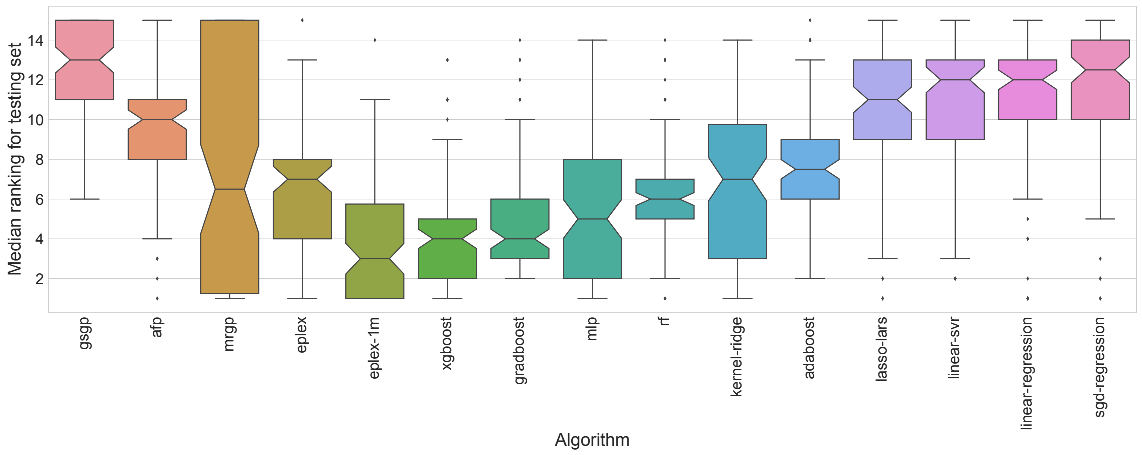

Fig. 3. Ranking of the performance of the algorithms based on the MSE score on testing datasets.

datasets was XGBoost. The top-performing GP method across all the datasets was MRGP and held the third place on average across training sets.

The test data results allow us to assess how well the algorithms handle generalization as well as their level of overfitting on training data. The relative performance of the methods changed noticeably when previously unseen data was used for evaluation. The results are presented in Fig. 3. The best performing method on average was EPLEX-1M. This GPSR method slightly outperformed XGBoost, which ended as the second best generalizing method across datasets. Gradient boosting was the third best method, and MLP finished in fourth place.

Several of the methods exhibit overfitting by changing ranking between the training and testing evaluations. Gradient boosting, for example, moves from first to third place. The performance of MRGP, which was one of the best regressors on the training data, also exhibits overfitting, resulting in a drop of its average ranking from 4th to 6th. MRGP's results also contained the highest variance in performance on test sets. GSGP exhibits the highest level of overfitting in terms rank changes, dropping from 8th to 13th. Conversely, several methods appear to generalize well, including EPLEX-1M (moving from a median ranking of 5 to 3) and Lasso (13 to 11).

We used the test set MSE scores to check for significant differences between methods across all datasets according to a Friedman test, which produces a p-value less than 2e-16, indicating significant differences. Post-hoc pairwise tests are then conducted and reported in Table 2. The large number of datasets provides higher statistical power than smaller experimental studies, leading to many p-values below 0.05. EPLEX-1M statistically outperforms the highest number of other methods (11), followed by XGBoost (9) and gradient boosting (7). We find that none of the comparisons between EPLEX-1M, XGBoost, gradient boosting and MLP are significantly different.

We now analyze the GP methods given equivalent numbers of fitness evaluations (AFP, MRGP, EPLEX, and GSGP). The results between MRGP and EPLEX show no significant difference. The only noted difference is that EPLEX significantly outperforms AFP, whereas MRGP does not. The three methods AFP, MRGP, and EPLEX all significantly outperform GSGP. Given more fitness evaluations, EPLEX-1M significantly outperforms all the other GP experiments, including EPLEX.

Table 2. Friedman Asymptotic General Symmetry Test. Bold indicates p <0.05.

	eplex-1m	xgboost	gradboost	mlp	rf	eplex	mrgp	kernel-ridge	adaboost	afp	lasso-lars	linear-svr	linear-regression	sgd-regression
xgboost	1	-	-	-	-	-	-	-	-	-	-	-	-	-
gradboost	0.9	1	-	-	-	-	-	-	-	-	-	-	-	-
mlp	0.2	0.6	1	-	-	-	-	-	-	-	-	-	-	-
rf	**0.003**	0.05	0.6	1	-	-	-	-	-	-	-	-	-	-
eplex	**0.001**	**0.02**	0.4	1	1	-	-	-	-	-	-	-	-	-
mrgp	**3e-07**	**2e-05**	**0.005**	0.2	0.9	1	-	-	-	-	-	-	-	-
kernel-ridge	**0.0007**	**0.02**	0.4	1	1	1	1	-	-	-	-	-	-	-
adaboost	**1e-07**	**4e-06**	**0.002**	0.1	0.8	0.9	1	0.9	-	-	-	-	-	-
afp	**3e-16**	**7e-14**	**4e-10**	**9e-06**	**0.0008**	**0.002**	0.3	**0.004**	0.5	-	-	-	-	-
lasso-lars	**0**	**0**	**2e-15**	**1e-11**	**1e-07**	**5e-07**	**0.002**	**6e-07**	**0.006**	1	-	-	-	-
linear-svr	**0**	**0**	**0**	**3e-13**	**2e-09**	**3e-08**	**0.0002**	**7e-08**	**0.0008**	0.8	1	-	-	-
linear-regression	**0**	**0**	**0**	**1e-14**	**7e-11**	**6e-10**	**5e-05**	**1e-09**	**0.0001**	0.5	1	1	-	-
sgd-regression	**0**	**0**	**0**	**0**	**1e-13**	**4e-12**	**1e-07**	**1e-12**	**5e-07**	0.07	0.9	1	1	-
gsgp	**0**	**0**	**0**	**0**	**0**	**0**	**2e-12**	**0**	**2e-11**	**0.0004**	0.1	0.4	0.7	1

The comparison of running times per training task is presented in Fig. 4. Three important considerations should be made when assessing these results. First, the experiment was conducted in a cluster environment. Second, each algorithm was run on a single thread for each dataset. Thus the easily parallelized algorithms (i.e., all GP-based methods and some ensemble tree methods) would likely show better relative performance in a multicore setting. Third, benchmarked algorithms were implemented using different programming languages. Thus, comparison of running times doesn't exclusively reflect the complexity of the methods.

Despite these considerations, it is worth noting how much additional computation time is required by the GP methods, which are one to three orders of magnitude slower than the nearest comparison. In terms of GP methods, MRGP runs the slowest, which may be partially due to its Java implementation (the other four GP methods use c++). EPLEX-1M is able to complete 10 times as many fitness evaluations in approximately the same time. The other three GP methods (GSGP, AFP, and EPLEX) show similar computation times. Among other ML methods, the ensemble tree methods and MLP are the slowest, and the linear methods are fastest, as expected.

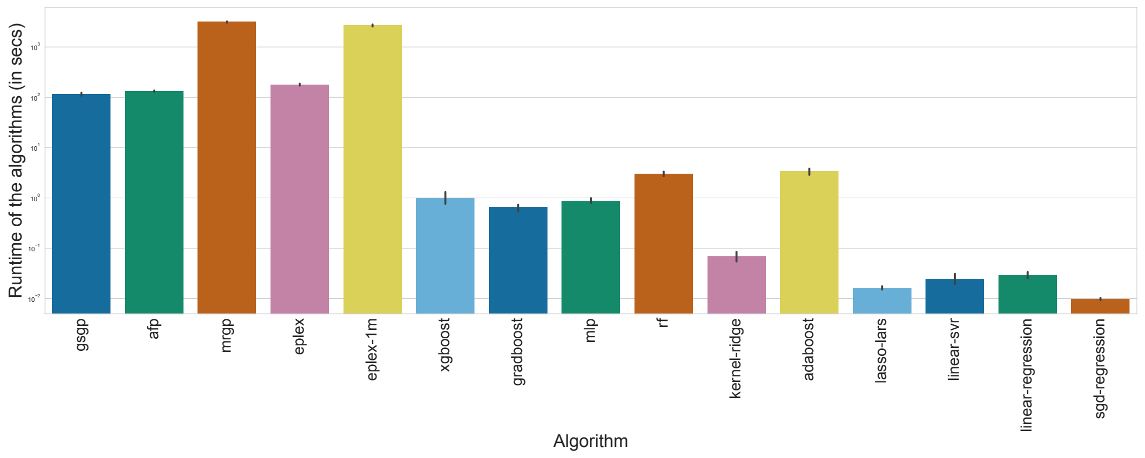

Fig. 4. Median running time of each of the algorithms (in seconds).

The most frequent settings of the parameters picked for the best model across all trials are presented in Table 3. We purposefully do not include Linear Regression in the table (run with the default values) or Kernel Ridge regression, for which multiple settings of input parameters performed comparably. It may be noted each GP-based method besides GSGP tended to favor large population sizes over larger numbers of generations. The optimal setting for crossover and mutation rates varied beteween methods.

Table 3. Most frequently chosen parameter settings based on 5-fold cross validation across all datasets.

Algorithm name	**Frequently best parameter settings**
gsgp	(‘g’=500, ‘max_len’=6, ‘popsize’=200, ‘rt_cross’=0.2, ‘rt_mut’=0.8)
afp	(‘g’=100, ‘max_len’=64, ‘popsize’=1000, ‘rt_cross’=0.8, ‘rt_mut’ 0.2)
mrgp	({‘g’=100, ‘pop_size’=1000} or the opposite; ’rt_cross’=0.2, ‘rt_mut’=0.8)
eplex	(‘g’=100, ‘max_len’=64, ‘popsize’=1000, ‘rt_cross’=0.8, ‘rt_mut’=0.2)
eplex-1m	(‘g’=500, ‘max_len’=100, ‘popsize’=2000, ‘rt_cross’=0.8, ‘rt_mut’=0.2)
xgboost	(‘’gamma’=0, ‘learning_rate’=0.01, ‘max_depth’=6, ‘n_estimators’=1000, ’subsample’=0.5)
gradboost	(‘max_features’=None, ‘min_weight_fraction_leaf’=0.0, ‘n_estimators’=1000)
mlp	(‘activation’=’logistic’, ‘learning_rate’= ‘constant’,‘solver’=‘lbfgs’)
rf	(‘max_features’=None, ‘min_weight_fraction_leaf’=0.0, ‘n_estimators’=1000)
adaboost	(‘learning_rate’=1.0, ‘n_estimators’=1000)
lasso-lars	(‘alpha’=‘0.001’)
linear-svr	(‘C’=0.1, ‘loss’=‘squared_epsilon_insensitive’)
sgd-regression	(‘alpha’=0.01, ‘penalty’=‘l1’)
linear-regression	(‘fit_intercept’ ‘True’)

4 CONCLUSIONS

In this paper we evaluated four recent GPSR methods in comparison to ten state-of-the-art ML methods on a set of 94 real-world regression problems. We consider hyper-parameter optimization for each method using nested cross-validation, and compare the methods in terms of the MSE they produce on training and testing sets, and their runtime. The analysis includes some interesting results. The most noteworthy finding is that a GPSR method (ϵ-lexicase selection implemented in ellyn), given 1 million fitness evaluations, achieves the best test set MSE ranking across all datasets and methods. Two of the GP-based methods, namely: EPLEX and MRGP, produce competitive results compared to state-of-the-art ML regression approaches. The downside of the GP-based methods is their computation complexity when run on a single thread, which contributes to much higher runtimes. Parallelism is likely to be a key factor in allowing GP-based approaches to become competitive with leading ML methods with respect to running times.

We also should note some shortcomings of this study that motivate further analysis. First, a guiding motivation for the use of GPSR is often its ability to produce legible symbolic models. Our analysis did not attempt to quantify the complexity of the models produced by any of the methods. An extension of this work could establish a standardized way of assessing this complexity, for example using the polynomial complexity method proposed by Vladislavleva et. al. [36]. Ultimately the relative value of explainability versus predictive power will depend on the application domain. Second, we have considered real world datasets for the source of our benchmarks. Simulation studies could also be used, and have the advantage of providing ground truth about the underlying process, as well as the ability to scale complexity or difficulty. It should also be noted that the datasets used for this study were of relatively small sizes (up to 1000 of instances). Future work should consider larger dataset sizes, but will come with a larger computational burden.

We have also limited our initial analysis to looking at bulk performance of algorithms over many datasets. Further analysis of these results should provide insight into the properties of datasets that make them amenable to, or difficult for, GP-based regression. Such an analysis can provide suggestions for new problem sub-types that may be of interest to the GP community.

We hope this study will provide the ML community with a data-driven sense of how state-of-the-art SR methods compare broadly to other popular ML approaches to regression.

SUPPLEMENTARY MATERIALS

Source code for our experiment can be found at the following url: https://github.com/EpistasisLab/regression-benchmark.

ACKNOWLEDGMENTS

The authors would like to thank Randal Olson, whose work helped motivate this study, and Weixuan Fu, who put together the regression benchmark resource. This research was supported in part by PL-Grid Infrastructure and by NIH grants LM010098 and AI116794.

REFERENCES

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

[2] J.C. Bongard and H. Lipson. 2005. Nonlinear System Identification Using Coevolution of Models and Tests. *IEEE Transactions on Evolutionary Computation* 9, 4 (Aug. 2005), 361–384. DOI:http://dx.doi.org/10.1109/TEVC.2005.850293

[3] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

[4] Mauro Castelli, Sara Silva, and Leonardo Vanneschi. 2015. A C++ framework for geometric semantic genetic programming. *Genetic Programming and Evolvable Machines* 16, 1 (March 2015), 73–81. DOI:http://dx.doi.org/10.1007/s10710-014-9218-0

[5] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM, 785–794.

[6] Grant Dick, Aysha P. Rimoni, and Peter A. Whigham. 2015. A Re-Examination of the Use of Genetic Programming on the Oral Bioavailability Problem. ACM Press, 1015–1022. DOI:http://dx.doi.org/10.1145/2739480.2754771

[7] Pedro Domingos. 2012. A few useful things to know about machine learning. *Commun. ACM* 55, 10 (2012), 78–87.

[8] Harris Drucker. 1997. Improving regressors using boosting techniques. In *ICML*, Vol. 97. 107–115.

[9] Chris Drummond and Nathalie Japkowicz. 2010. Warning: statistical benchmarking is addictive. Kicking the habit in machine learning. *Journal of Experimental & Theoretical Artificial Intelligence* 22, 1 (March 2010), 67–80. DOI:http://dx.doi.org/10.1080/09528130903010295

[10] Bradley Efron, Trevor Hastie, Iain Johnstone, Robert Tibshirani, and others. 2004. Least angle regression. *The Annals of statistics* 32, 2 (2004), 407–499.

[11] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. 2014. Do we need hundreds of classifiers to solve real world classification problems. *J. Mach. Learn. Res* 15, 1 (2014), 3133–3181.

[12] Yoav Freund and Robert E Schapire. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences* 55, 1 (1997), 119–139.

[13] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[14] Geoffrey E Hinton. 1989. Connectionist Learning Procedures. *Artificial Intelligence* 40 (1989), 185–234.

[15] Gregory S Hornby. 2006. ALPS: the age-layered population structure for reducing the problem of premature convergence. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. ACM, 815–822. DOI:http://dx.doi.org/10.1145/1143997.1144142

[16] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[17] Michael F. Korns. 2011. Accuracy in symbolic regression. In *Genetic Programming Theory and Practice IX*. Springer, 129–151. http://link.springer.com/chapter/10.1007/978-1-4614-1770-5_8

[18] William La Cava, Kourosh Danai, and Lee Spector. 2016. Inference of compact nonlinear dynamic models by epigenetic local search. *Engineering Applications of Artificial Intelligence* 55 (Oct. 2016), 292–306. DOI:http://dx.doi.org/10.1016/j.engappai.2016.07.004

[19] William La Cava, Kourosh Danai, Lee Spector, Paul Fleming, Alan Wright, and Matthew Lackner. 2016. Automatic identification of wind turbine models using evolutionary multiobjective optimization. *Renewable Energy* 87, Part 2 (March 2016), 892–902. DOI: http://dx.doi.org/10.1016/j.renene.2015.09.068

[20] William La Cava, Lee Spector, and Kourosh Danai. 2016. Epsilon-Lexicase Selection for Regression. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016 (GECCO '16)*. ACM, New York, NY, USA, 741–748. DOI: http://dx.doi.org/10.1145/2908812.2908898

[21] James McDermott, David R. White, Sean Luke, Luca Manzoni, Mauro Castelli, Leonardo Vanneschi, Wojciech Jaskowski, Krzysztof Krawiec, Robin Harper, and Kenneth De Jong. 2012. Genetic programming needs better benchmarks. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*. ACM, 791–798. http://dl.acm.org/citation.cfm?id=2330273

[22] Alberto Moraglio, Krzysztof Krawiec, and Colin G. Johnson. 2012. Geometric semantic genetic programming. In *Parallel Problem Solving from Nature-PPSN XII*. Springer, 21–31. http://link.springer.com/chapter/10.1007/978-3-642-32937-1_3

[23] Quang Uy Nguyen, Tuan Anh Pham, Xuan Hoai Nguyen, and James McDermott. 2015. Subtree semantic geometric crossover for genetic programming. *Genetic Programming and Evolvable Machines* (Oct. 2015), 1–29. DOI: http://dx.doi.org/10.1007/s10710-015-9253-5

[24] Randal S. Olson, William La Cava, Zairah Mustahsan, Akshay Varik, and Jason H. Moore. 2017. Data-driven Advice for Applying Machine Learning to Bioinformatics Problems. In *Pacific Symposium on Biocomputing (PSB)*. http://arxiv.org/abs/1708.05070 arXiv: 1708.05070.

[25] Randal S. Olson, William La Cava, Patryk Orzechowski, Ryan J. Urbanowicz, and Jason H. Moore. 2017. PMLB: A Large Benchmark Suite for Machine Learning Evaluation and Comparison. *BioData Mining* (2017). https://arxiv.org/abs/1703.00512 arXiv preprint arXiv:1703.00512.

[26] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, and others. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12, Oct (2011), 2825–2830.

[27] Christian Robert. 2014. Machine learning, a probabilistic perspective. (2014).

[28] Michael Schmidt and Hod Lipson. 2009. Distilling free-form natural laws from experimental data. *Science* 324, 5923 (2009), 81–85. http://www.sciencemag.org/content/324/5923/81.short

[29] Michael Schmidt and Hod Lipson. 2011. Age-fitness pareto optimization. In *Genetic Programming Theory and Practice VIII*. Springer, 129–146. http://link.springer.com/chapter/10.1007/978-1-4419-7747-2_8

[30] Michael D Schmidt, Ravishankar R Vallabhajosyula, Jerry W Jenkins, Jonathan E Hood, Abhishek S Soni, John P Wikswo, and Hod Lipson. 2011. Automated refinement and inference of analytical models for metabolic networks. *Physical Biology* 8, 5 (Oct. 2011), 055011. DOI: http://dx.doi.org/10.1088/1478-3975/8/5/055011

[31] Alex J Smola and Bernhard Schölkopf. 2004. A tutorial on support vector regression. *Statistics and computing* 14, 3 (2004), 199–222.

[32] Lee Spector. 2012. Assessment of problem modality by differential performance of lexicase selection in genetic programming: a preliminary report. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion*. 401–408. http://dl.acm.org/citation.cfm?id=2330846

[33] Karolina Stanislawska, Krzysztof Krawiec, and Zbigniew W. Kundzewicz. 2012. Modeling global temperature changes with genetic programming. *Computers & Mathematics with Applications* 64, 12 (Dec. 2012), 3717–3728. DOI: http://dx.doi.org/10.1016/j.camwa.2012.02.049

[34] Robert Tibshirani. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)* (1996), 267–288.

[35] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. 2014. OpenML: Networked Science in Machine Learning. *SIGKDD Explor. Newsl.* 15, 2 (June 2014), 49–60. DOI: http://dx.doi.org/10.1145/2641190.2641198

[36] E.J. Vladislavleva, G.F. Smits, and D. den Hertog. 2009. Order of Nonlinearity as a Complexity Measure for Models Generated by Symbolic Regression via Pareto Genetic Programming. *IEEE Transactions on Evolutionary Computation* 13, 2 (2009), 333–349. DOI: http://dx.doi.org/10.1109/TEVC.2008.926486

[37] Jan Žegklitz and Petr Pošík. 2017. Symbolic Regression Algorithms with Built-in Linear Regression. *arXiv:1701.03641 [cs]* (Jan. 2017). http://arxiv.org/abs/1701.03641 arXiv: 1701.03641.

[38] David R. White, James McDermott, Mauro Castelli, Luca Manzoni, Brian W. Goldman, Gabriel Kronberger, Wojciech Jaśkowski, Una-May O'Reilly, and Sean Luke. 2012. Better GP benchmarks: community survey results and proposals. *Genetic Programming and Evolvable Machines* 14, 1 (Dec. 2012), 3–29. DOI: http://dx.doi.org/10.1007/s10710-012-9177-2