Published in final edited form as:
Genet Evol Comput Conf. 2019 July ; 2019: 1056–1064. doi:10.1145/3321707.3321776.

Semantic variation operators for multidimensional genetic programming

William La Cava[*],
Jason H. Moore
University of Pennsylvania, Philadelphia, PA

Abstract

Multidimensional genetic programming represents candidate solutions as sets of programs, and thereby provides an interesting framework for exploiting building block identification. Towards this goal, we investigate the use of machine learning as a way to bias which components of programs are promoted, and propose two semantic operators to choose where useful building blocks are placed during crossover. A forward stagewise crossover operator we propose leads to significant improvements on a set of regression problems, and produces state-of-the-art results in a large benchmark study. We discuss this architecture and others in terms of their propensity for allowing heuristic search to utilize information during the evolutionary process. Finally, we look at the collinearity and complexity of the data representations that result from these architectures, with a view towards disentangling factors of variation in application.

Keywords

representation learning; feature construction; variation; regression

1 INTRODUCTION

A central theme in genetic programming (GP) is how to identify, propagate, and properly compose the components of programs that contribute to good solutions. In the context of classification and regression, these building blocks fill the role of "feature engineering". That is to say, building blocks of GP solutions are meant to explain the underlying factors of variation that produce the observed response. The task of optimizing a set of explanatory features for a problem is known as *representation learning*, especially in the larger machine learning (ML) community [3]. Representation learning is a fundamental challenge in ML due to its computational complexity and the role the representation plays in

[*]Corresponding Author: lacava@upenn.edu.

[7]SUPPLEMENTARY MATERIAL
Code to reproduce the experiments can be found at http://github.com/lacava/gecco_2019.

model accuracy and interpretation. Interestingly, a variant of GP known as multidimensional GP (MGP) makes this relationship between building block discovery and representation learning explicit by optimizing a set of programs, each of which is an independent feature in the ML model. Our goal in this paper is to introduce semantic variation methods to MGP, with the goal of improving the representations it produces.

What makes a representation good? At the minimum, a good representation produces a model with better generalization than a model trained only on the raw data attributes. In addition, a good representation teases apart the factors of variation in the data into independent components. Finally, an ideal representation is succinct so as to promote intelligibility. In other words, a representation should only have as many features as there are independent factors in the process. Our discussion centers around these three motivations.

In the following section, we attempt to summarize the large body of work concerning feature construction / representation learning in GP, especially those methods that use ML to promote building blocks. This provides context for the MGP family of methods. We then describe our main contribution: the proposed methods of crossover in Section 3. We conduct an experiment at first on 8 regression problems, considering full hyperparameter tuning, and analyze the representations that are produced with and without the new crossover methods. Finally, we benchmark the new methods against many ML and GP methods on more than 100 open source regression problems. We find that the new methods of crossover lead to state-of-the-art results for regression. Our discussion points to further directions for improving representation quality within this framework.

2 BACKGROUND

Feature construction / representation learning has been a consistent theme in the GP community and has been studied with various architectures. Without major changes, GP can be applied to the task of identifying single features for regression and classification (or multiple features in the multiclass case [30]), and this approach has been explored in several works, summarized in [27]. These filter approaches generally use an information-theoretic measure to determine how good a program is likely to be as a feature. Optimizing single features requires basically no changes to GP’s methodology, but lacks the power to optimize features for the multivariate context in which they are typically used.

Another approach that has been studied is to treat each individual in the population as a feature, and to optimize an ensemble model of the entire population [2, 6, 20, 21, 40]. Only a single regression model is trained per generation, which demands minimal overhead. However, it is not well known how to properly select and vary features evolved by such a process. Since each individual is a feature, its fitness is heavily dependent on the current population. Furthermore, a desirable set of features should be essentially orthogonal in order to create a well-conditioned representation, and convergent evolutionary processes aim in the opposite direction. To overcome issues of collinearity and a convergent search process, the following ideas have been proposed. In evolutionary feature synthesis (EFS) [2], features are selected proportionally to their coefficient in a regularized linear model; in order to prevent multicollinearity, correlation thresholds are implemented during variation to keep children different from their parents. In the feature engineering wrapper (FEW) [20, 21], multicollinearity is selected against by using a survival version of ϵ-lexicase selection to choose features. In Kaizen GP [6, 40], individuals are only added to the model if they pass a significance test, in a hill climbing fashion. Another option is to not use an evolutionary updating scheme at all, but rather to create a large set of random features and fit an ML model to this, as in FFX [25]. More recently, Vanneschi et. al. explored one step linear combinations of random programs [39], experimentally showing that they often lead to overfitting.

Rather than building a model from the entire population, one could apply an ML method to the entire program trace as a means of identifying building blocks [17]. Multiple regression GP (MRGP) [1] defines a program's behavior as the Lasso [37] estimate generated over the entire program's trace. One downside of this approach is the likely presence of highly correlated features in the program trace, leading to an ill-conditioned regression matrix. In a similar vein to MRGP, Behavioral GP [18] extracts information from the entire program trace, this time using a decision tree algorithm to identify important building blocks, which are stored in an archive for re-use. In both algorithms, the key insight is to use ML with program traces to undo the complex masking effect that program execution has on the behavior of building blocks that are downstream from other operations in the program (for further discussion on the topic of program traces see [16]).

MGP is a framework that is, in some sense, in between the ensemble techniques and the program trace techniques described above. Programs are represented as sets of separate subprograms, usually trees. Unlike population-wide models, the fitness of each individual is directly related to its model predictions, and individuals in the population benefit from typical evolutionary optimization processes. Unlike program trace-based methods, by using multi-output programs, MGP exposes *independent components* of the total program behavior to the ML process that produces the model. As a result, building blocks are easier to isolate and share among the population in direct ways.

Examples of MGP include Krawiec's method [15], M2GP [13], M3GP [28], e-M3GP [36], M4GP [22], and FEAT [23]. In all of these methods, individuals in the population produce a set of corresponding outputs that are then fed into a deterministic ML method to produce the program's regression or classification estimates. In the case of M2GP, M3GP, and M4GP, classification proceeds using a nearest centroid classifier [38], whereas linear regression methods are used for regression with M3GP [29] and FEAT. Although a number of methods have been proposed in the MGP paradigm, they have not made much use of the semantics of independent building blocks in each program that this architecture creates. An exception is [23], in which the authors use the coefficient magnitude to weight probabilities of mutation. The main contribution of this study is the development of semantic crossover schemes to leverage the architecture of MGP to a larger degree than these previous studies.

3 METHODS

In this paper we focus on the task of regression, with the goal of building a predictive model $\hat{y}(\mathrm{x})$ using N paired examples $\mathcal{T} = \{(\mathrm{x}_i, y_i)\}_{i=1}^{N}$. The regression model $\hat{y}(\mathrm{x})$ associates the inputs $\mathrm{x} \in \mathbb{R}^d$ with a real-valued output $y \in \mathbb{R}$. The goal of feature engineering / representation learning is to find a new representation of x via a *m*-dimensional feature mapping $\phi(\mathrm{x}): \mathbb{R}^d \to \mathbb{R}^m$, such that the model $\hat{y}(\phi(\mathrm{x}))$ outperforms the model $\hat{y}(\mathrm{x})$ by some pre-defined metric (see above).

In MGP, each individual in the population is a candidate representation, $\phi(\mathrm{x})$, consisting of a set of programs $[\phi_1, \ldots, \phi_m]$. As an example, the individual

$$[(+(x_1)(x_2)),\ (\cos(x_3)),\ (\exp(\text{cube}\,(x_1)))]$$

would encode a representation with three features: $(x_1 + x_2)$, $\cos(x_3)$, and $\exp\left(x_1^3\right)$. Throughout the paper, we refer to these subprograms ϕ as *features*, and use the word *attribute* to refer to the independent variables in x.

MGP methods share this representation in common, and differ in terms of 1) the ML method used to generate the model prediction, i.e. $\hat{y}(\phi)$, 2) the crossover and mutation operators used, and 3) the selection process used. The crossover operators proposed in this section will work with any MGP method, but are designed with a linear ML pairing in mind.

3.1 Feature Engineering Automation Tool

We study a recent MGP method named the Feature Engineering Automation Tool (FEAT) [23], in which candidate features are parameterized by weights, θ, and used to fit a linear model

$$\hat{y} = \sum_{i=1}^{m} \beta_i \phi_i(\mathrm{x}, \theta) \tag{1}$$

The coefficients $[\beta_1, \ldots, \beta_m]$ are determined using ridge regression [12]. Note that each ϕ is normalized to zero mean, unit variance before ridge regression is applied. The parameters θ are attached to the edges of differentiable operators and updated each generation via gradient descent. The fitness of each individual in FEAT is its mean squared error (MSE) on the training set.

Feedback.—In order to promote building blocks, FEAT uses feedback from the ML process to bias the variation step. In a nutshell, the probability of a feature in ϕ being mutated or replaced by crossover is inversely related the magnitude of its coefficient β in Eqn. 1. Let $\tilde{\beta}_i(n) = |\beta_i| / \sum_i^m |\beta_i|$. The normalized coefficient magnitudes $\tilde{\beta} \in [0, 1]$ are used to define softmax-normalized probabilities. The probability of mutation for feature *i* in program *n* is denoted $PM_i(n)$, and defined as follows:

$$\begin{aligned} s_i(n) &= \exp\left(1-\tilde{\beta}_i\right) / \sum_i^m \exp\left(1-\tilde{\beta}_i\right) \\ PM_i(n) &= \gamma s_i(n) + (1-\gamma)\frac{1}{m} \end{aligned} \tag{2}$$

Here, γ is a parameter that controls the amount of feedback from the weights that is used to bias the selection of feature i for mutation. In our experiments, we tune γ, and also test whether the softmax normalization of $s_i(n)$ is useful.

Selection.—In the initial work introducing FEAT [23], the authors compared several optimization schemes, including NSGA-II, simulated annealing, random search, ϵ-lexicase selection [24], and a hybrid of ϵ-lexicase selection with NSGA-II's survival scheme. The final combination achieved the best results in the analysis, and is used in our work here. The original work attempted to address the issue of disentanglement by measuring the multicollinearity of representations and setting this metric as an additional objective during survival. However, none of the objectives tried resulted in less correlated final representations, and actually made them slightly worse. One of the goals of this paper is to explore a second hypothesis, that disentangled representations can be encouraged through the use of semantic variation operators, rather than through additional objectives.

Variation.—Semantic variation operators have not yet been proposed for MGP frameworks. The most recent MGP techniques (M3GP, M4GP and FEAT) adopt special variation operators that vary programs at the feature level. Feature crossover (called "root crossover" in [28]) swaps features between two parent representations. Feature mutation may delete a feature or add a random feature. M3GP, M4GP and FEAT also use standard subtree crossover and point mutation operators, and each of these operators occur with equal probability. In FEAT, however, the probabilities of each feature being chosen for mutation or crossover is determined by Eqn. 2.

In the following section, we describe two new semantic crossover operators for MGP that attempt to maintain orthogonality in the representations while moving towards models with lower residuals.

3.2 Semantic Crossover

The following two crossover methods are called *semantic* because they use information about the program's outputs to determine the recombination that occurs to produce a child from two parents. Both operators are based on the following observations. We have two parent representations, $\boldsymbol{\phi}_{p1}$ and $\boldsymbol{\phi}_{p2}$, with corresponding model outputs $\hat{y}_{p1}$ and $\hat{y}_{p2}$ that are linear combinations of their respective representations, as in Eqn. 1. We want to produce the best combination of $\boldsymbol{\phi}_{p1}$ and $\boldsymbol{\phi}_{p2}$ for the child representation $\boldsymbol{\phi}_c$. Basically we can treat this as a feature selection problem, where we have features $\phi_A = \phi_{p1} \cup \phi_{p2}$ and we want to pick the best. On one hand we could simply concatenate the feature sets, and generate a new model $\hat{y}(\phi_A)$, which is the linear model fit to all features of both parents. This approach would lead to exponential growth in offspring, which would run against our goal of lowering complexity.

In lieu of that approach, we propose here what are essentially regularized versions of semantic crossover that constrain the number of features in the offspring to be of equal cardinality to ϕ_{p1}, i.e. $|\phi_c| = |\phi_{p1}|$. The first operator, best residual fit crossover (ResXO), chooses a feature from ϕ_{p1} to be replaced, and then chooses the feature in ϕ_{p2} that best approximates the residual of the model after removing this feature. The second operator, stagewise crossover (StageXO), uses forward stagewise regression [14] as a feature selection method to iteratively construct the offspring.

3.3 Best residual fit crossover (ResXO)

Given parents $p1$ and $p2$, ResXO swaps a feature in $p1$ with the feature in $p2$ that most closely approximates the residual error of $p1$ with the selected feature removed. The child representation is denoted as ϕ_c. The steps are as follows:

1. pick ϕ_d from ϕ_{p1} using probabilities given by Eqn. 2.
2. calculate the residual of $p1$ without ϕ_d:

$$r = y - \hat{y}_{p1} - \beta_d \phi_d$$

3. choose $\phi*$ from ϕ_{p2}, which is the feature most correlated with r.
4. $\phi_c = \phi_{p1}$ with ϕ_d replaced by $\phi*$.

ResXO is a semantic backpropagation operator [7, 11, 33], since it seeks to replace a component of the parent program with a subprogram most closely matching the desired semantics, given by r. Within the MGP framework, this backpropagation is very simple, and does not require complex inversion operations to be introduced. We expect that ResXO will also lead to lower correlations between features in ϕ_c than in ϕ_{p1}. To understand why, consider that

$$r = y - \sum_{\phi_i \in \phi_{p1} \setminus \phi_d} \beta_i \phi_i$$

Therefore r should have low correlation with the rest of the $p1$'s representation. Assuming the replacement feature from ϕ_{p2} closely matches r, it should also be uncorrelated with $\{\phi_{p1} \setminus \phi_d\}$. Note that ResXO may produce an individual with higher squared error than its parents, since ϕ_d may be more correlated with r than $\phi*$.

3.4 Forward stagewise crossover (StageXO)

Rather than restricting crossover to the replacement of a single feature, the crossover operator can be used to compile the set of features that iteratively reduce the target error using a forward stagewise crossover method we call StageXO. The procedure is as follows:

```
1. set the initial residual equal to the target: $r = y$. Center means around zero for all $\phi$.
2. set $\phi_A$ to be all subprograms in $\phi_{p1}$ and $\phi_{p2}$.
3. while $|\phi_c| < |\phi_{p1}|$:
    a. pick $\phi*$ from $\phi_A$ which is most correlated with $r$.
    b. compute the least squares coefficient $b$ for $\phi*$ fit to $r$.
    c. update $r = r - b\phi*$
    d. add $\phi*$ to $\phi_c$.
    e. remove $\phi*$ from $\phi_A$.
```

Unlike feature selection methods like forward/backward stepwise selection, forward stagewise selection only calculates the weight of a single feature at a time, and is thus more lightweight. The downside of this approach in the context of regression is that it generally takes more iterations to reach the least squares coefficients of the complete model [10]. In our case this is unimportant, since we are only interested in quickly choosing the most important features, which are then used to fit a multiple linear regression model. We expect the child representation returned by StageXO to contain uncorrelated features since the residual is updated each iteration to remove the portion of the response explained by previous features.

Forward stagewise regression, and therefore the StageXO operator, is closely related to boosting [9]. In both cases the residual is iteratively reduced by adding model components (weak learners in the case of boosting, and features/building blocks in our case). The relationship between forward stagewise regression, boosting, and regularized linear models is expounded upon in [10]. The stagewise additive modeling paradigm is also used by a recent GP technique called Wave [26], in which GP runs are iteratively trained on residuals of previous runs. The insight here is that the unique representation of programs in MGP allows the same general methodology to be exploited for combining partial solutions during crossover, rather than as a post-run ensemble method.

4 EXPERIMENT

Our experiment consists of two stages. First, we conduct an extensive study of FEAT with and without the semantic crossover operators introduced in Section 3. In this study we vary the hyperparameters related to variation and analyze the results in detail for 8 regression problems. In the second study, we apply FEAT, FEATResXO, and FEATStageXO to more than 100 problems from the PMLB regression benchmark [32]. These variants are compared to state-of-the-art symbolic regression and ML methods.

4.1 Hyperparameter study

Despite several MGP methods having been proposed, there has not been a systematic study of the effect of variation operators on the performance of this family of methods. To fill this gap, and to properly analyze the new methods introduced in this paper, we performed a grid search of variation hyperparameters on 8 regression problems. The hyperparameters that were varied are shown in Table 1.

Feedback softmax normalization refers to the softmax transformation in Eqn. 2; we tested for whether this normalization, which assumes a multinomial distribution of probabilities, was useful. The eight comparison problems are listed in Table 2.

4.2 Benchmark comparison

In the second study, we compared FEAT with each crossover variant to 15 other methods: 5 GP methods [1, 4, 19, 35] and 10 ML methods from scikit-learn [34]. The 5 GP methods we compared to are:

- Geometric Semantic GP (GSGP) [4]
- MRGP [1]
- Age-fitness Pareto Optimization (AFP) [35]
- ϵ-lexicase selection (EPLEX) [19]
- ϵ-lexicase selection with 1 million evaluations (EPLEX-1M) [19]

These methods were benchmarked on 94 open-source datasets collected in the Penn ML Benchmark [31]. We used results from Orzechowski et. al.'s benchmark analysis [32] as a comparison, and followed the same validation procedure. Each comparison method underwent hyperparameter tuning using 5-fold cross validation on a 75% split of the training set, and was then tested on a 25% test fold. The hyperparameters are detailed in Table 1 of the original work [32]. This process was repeated for 10 trials. GP methods were given 100,000 evaluations, apart from EPLEX-1M which used 1 million. For FEAT, we did not re-tune the hyperparameters, instead using the values determined from the hyperparameter tuning experiment.

We also extended the PMLB comparison to larger datasets because we were interested in 1) how FEAT handled larger datasets (the original study was restricted to smaller datasets, c.f. Fig. 1 of [32]) and also how the size of the final models compared to other top-ranking algorithms such as XGBoost [5] and multilayer perceptron (MLP). The extended analysis included 111 datasets, whose properties are shown in Fig. 1. These datasets are used to evaluate the complexity of the final models.

4.3 Metrics

As mentioned earlier, we consider there to be three over-arching goals when learning a representation. The first is that $\phi(x)$ leads to a model with a low generalization error. To measure this, we compare the mean squared error (MSE) and coefficient of determination (R^2) of each model output on the test set. We also wish to minimize the complexity of the representation. To measure the complexity of solutions in FEAT, we count the total number of nodes in the final representation. For comparison to XGBoost, we count the number of nodes in the trees, and for comparison to MLP, we count the number of nodes in the network. Finally, we want a representation that is "disentangled", meaning that each feature of $\boldsymbol{\phi}$ is as orthogonal to the others as possible. One such measure is the pairwise Pearson correlations of features in $\boldsymbol{\phi}$, written as

$$Corr(\phi) = \frac{1}{N(N-1)} \sum_{\phi_i, \phi_j \in \phi, i \neq j} \left(\frac{\mathrm{cov}(\phi_i, \phi_j)}{\sigma(\phi_i)\sigma(\phi_j)} \right)^2 \tag{3}$$

We use this equation to compare the final representations across selection methods.

5 RESULTS

The hyperparameter tuning results are presented first. In addition to test score reporting, we plot various views of the data with respect to different hyperparameters, and also look at representation correlations in the resultant models and statistical comparisons. In the subsequent section, the PMLB comparison results are shown, including score comparisons, runtime comparisons, statistical tests and comparisons of the final model sizes for FEAT, XGBoost, and MLP learners.

5.1 Hyperparameter tuning

Prediction comparisons for each crossover method are shown in Fig. 2 for the 8 tuning problems. The plot shows the mean test fold R^2 value for the tuned estimator, summarized across trials. In general one can see that StageXO produces the most accurate results, followed by ResXO. Across the 8 problems, StageXO significantly outperforms standard crossover (p <0.035); the pairwise statistical comparisons are given in Table 3.

We also looked at the correlation of the representations produced by the different crossover methods, shown in Fig. 3. We confirmed our hypotheses that ResXO and StageXO would produce less correlated representations than the traditional crossover operator.

The best values for each tuned parameter is shown in Table 4. We found that softmax normalization did not improve the feedback probabilities. Across problems, the best crossover/mutation fraction was found to be 0.75 (Fig. 4), with a feature crossover rate of 0.75 for Feat and 0.5 for ResXO and StageXO. The best feedback value was problem dependent, as shown in Fig. 5. Since the feedback essentially controls the amount of exploration versus exploitation, it stands to reason that the ideal setting of this parameter would be problem dependent. Feedback levels of 0.25 were best for FEAT and FEATStageXO, and no feedback was best for FEATResXO. For the ResXO operator, this corresponds to choosing the feature to swap out of the parent at random.

5.2 Benchmark comparison

The comparisons of FEAT to 15 other methods is shown in Fig. 6. Across problems, FEATStageXO achieves a nearly identical ranking to EPLEX-1M, which is ϵ-lexicase selection run for 1 million evaluations. In this case, FEATStageXO achieves these similar results using 100,000 evaluations. However, the additional complexity of fitting ML models to each individual makes the evaluation of each individual in FEAT more costly. The wall clock times shown in Fig. 7 reflect this, as the FEAT wall clock times sit somewhere between the methods that ran for 100,000 evaluations (GSGP, AFP, MRGP, EPLEX) and EPLEX-1M.

A Friedman test of the MSE rankings across problems indicates significant differences. Table 5 shows post-hoc pairwise Wilcoxon signed rank tests of the results. FEAT, FEATResXO, and FEATStageXO all significantly outperform the other GP-based methods run for 100,00 evaluations. There is no significant difference found between the FEAT variants, EPLEX-1M, XGBoost, GradBoost, or MLP. The FEAT variants significantly outperform all other ML methods across the benchmark problems.

We extend the comparison of FEAT to XGBoost, MLP and ElasticNet on 111 of the datasets in PMLB in order to evaluate the complexity of the final representations. The size comparisons are shown in Fig. 8. In general FEAT produces representations about 1.5 orders of magnitude smaller than XGBoost and MLP. We found that StageXO led to slightly larger models than Feat or FeatResXO.

6 DISCUSSION & CONCLUSION

This paper proposes semantic crossover operators for multidimensional genetic programming. We contend that MGP provides an interesting framework for developing semantic variation methods, due to the unique way that program semantics are easily teased apart in the multi-output architecture. The most successful of these is forward stagewise crossover (StageXO), which mimics forward stagewise selection to incrementally build an offspring from the representations of its parents. A lightweight version of semantic backpropagation crossover, ResXO, is also proposed. StageXO achieves better performance on 8 regression problems than naive crossover operators, taking into account hyperparameter tuning. We find that the resultant representations are also less “entangled”, i.e. less correlated when using these crossover methods.

We consider this to be a first foray into semantic methods for MGP, and consider the potential for more powerful techniques to be high. One can imagine many more semantic operators that take advantage of this architecture. Three extensions come to mind. The first is to consider the features of many programs during crossover, rather than just two parents, as is done with population wide semantic crossover [39], behavioral GP, or in recent work by Fine et. al. [8]. The second extension would be to explicitly design variation operators that improve the condition of the representation, for example by pruning highly correlated features. A third extension is to implement smarter termination conditions for stagewise crossover that take into account the fitness of the parents or an error tolerance.

The current methods appear to be competitive with state-of-the-art regression techniques. In particular, FEAT is able to achieve top ranking results with less computational complexity than the previous top GP method on the PMLB benchmark. It is also able to produce much smaller representations than XGBoost and MLP. These promising results should motivate further research within the MGP framework.

ACKNOWLEDGMENTS

This work was supported by NIH grants AI116794 and LM012601, as well as the PA CURE grant from the Pennsylvania Department of Health. Special thanks to Tilak Raj Singh and members of the Computational Genetics Lab.

REFERENCES

[1]. Arnaldo Ignacio, Krawiec Krzysztof, and O'Reilly Una-May. 2014. Multiple regression genetic programming. In Proceedings of the 2014 conference on Genetic and evolutionary computation. ACM Press, 879–886. 10.1145/2576768.2598291

[2]. Arnaldo Ignacio, O'Reilly Una-May, and Veeramachaneni Kalyan. 2015. Building Predictive Models via Feature Synthesis. ACM Press, 983–990. 10.1145/2739480.2754693

[3]. Bengio Yoshua, Courville Aaron, and Vincent Pascal. 2013. Representation learning: A review and new perspectives. IEEE transactions on pattern analysis and machine intelligence 35, 8 (2013), 1798–1828. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6472238 [PubMed: 23787338]

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

[5]. Chen Tianqi and Guestrin Carlos. 2016. XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16). ACM, New York, NY, USA, 785–794. 10.1145/2939672.2939785

[6]. De Melo Vinícius Veloso. 2014. Kaizen programming. ACM Press, 895–902. 10.1145/2576768.2598264

[7]. Ffrancon Robyn and Schoenauer Marc. 2015. Memetic Semantic Genetic Programming. ACM Press, 1023–1030. 10.1145/2739480.2754697

[8]. Fine Steven B., Hemberg Erik, Krawiec Krzysztof, and O'Reilly Una-May. 2018. Exploiting Subprograms in Genetic Programming. In Genetic Programming Theory and Practice XV (Genetic and Evolutionary Computation), Banzhaf Wolfgang, Olson Randal S., Tozier William, and Riolo Rick (Eds.). Springer International Publishing, 1–16.

[9]. Freund Yoav and Schapire Robert E. 1995. A desicion-theoretic generalization of on-line learning and an application to boosting. In Computational learning theory. Springer, 23–37. http://link.springer.com/chapter/10.1007/3-540-59119-2_166

[10]. Friedman Jerome, Hastie Trevor, and Tibshirani Robert. 2001. The elements of statistical learning. Vol. 1. Springer series in statistics Springer, Berlin. http://statweb.stanford.edu/~tibs/book/preface.ps

[11]. Graff Mario, Tellez Eric Sadit, Villaseñor Elio, and Miranda Sabino. 2015. Semantic Genetic Programming Operators Based on Projections in the Phenotype Space. (2015), 13.

[12]. Hoerl Arthur E. and Kennard Robert W. 1970. Ridge regression: Biased estimation for nonorthogonal problems. Technometrics 12, 1 (1970), 55–67.

[13]. Ingalalli Vijay, Silva Sara, Castelli Mauro, and Vanneschi Leonardo. 2014. A Multi-dimensional Genetic Programming Approach for Multi-class Classification Problems. In Genetic Programming. Springer, 48–60. http://link.springer.com/chapter/10.1007/978-3-662-44303-3_5

[14]. James Gareth, Witten Daniela, Hastie Trevor, and Tibshirani Robert. 2013. An Introduction to Statistical Learning. Springer Texts in Statistics, Vol. 103. Springer New York, New York, NY. 10.1007/978-1-4614-7138-7

[15]. Krawiec Krzysztof. 2002. Genetic programming-based construction of features for machine learning and knowledge discovery tasks. Genetic Programming and Evolvable Machines 3, 4 (2002), 329–343. http://link.springer.com/article/10.1023/A:1020984725014

[16]. Krawiec Krzysztof. 2012. On relationships between semantic diversity, complexity and modularity of programming tasks. In Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference. ACM, 783–790. http://dl.acm.org/citation.cfm?id=2330272

[17]. Krawiec Krzysztof. 2016. Behavioral program synthesis with genetic programming. Vol. 618. Springer.

[18]. Krawiec Krzysztof and O'Reilly Una-May. 2014. Behavioral programming: a broader and more detailed take on semantic GP. In Proceedings of the 2014 conference on Genetic and evolutionary computation. ACM Press, 935–942. 10.1145/2576768.2598288

[19]. Cava William La, Helmuth Thomas, Spector Lee, and Moore Jason H. 2018. A probabilistic and multi-objective analysis of lexicase selection and ε -lexicase selection. Evolutionary Computation (May 2018), 1–28. 10.1162/evco_a_00224

[20]. Cava William La and Moore Jason. 2017. A General Feature Engineering Wrapper for Machine Learning Using \epsilon -Lexicase Survival. In Genetic Programming. Springer, Cham, 80–95. 10.1007/978-3-319-55696-3_6

[21]. Cava William La and Moore Jason H. 2017. Ensemble representation learning: an analysis of fitness and survival for wrapper-based genetic programming methods. In GECCO '17: Proceedings of the 2017 Genetic and Evolutionary Computation Conference. ACM, Berlin, Germany, 961–968. 10.1145/3071178.3071215

[22]. Cava William La, Silva Sara, Danai Kourosh, Spector Lee, Vanneschi Leonardo, and Moore Jason H. 2018. Multidimensional genetic programming for multiclass classification. Swarm and Evolutionary Computation (April 2018). 10.1016/j.swevo.2018.03.015

[23]. Cava William La, Singh Tilak Raj, Taggart James, Suri Srinivas, and Moore Jason H. 2019. Learning concise representations for regression by evolving networks of trees. In International Conference on Learning Representations (ICLR). https://arxiv.org/abs/1807.00981 In Press.

[24]. Cava William La, Spector Lee, and Danai Kourosh. 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. 10.1145/2908812.2908898

[25]. McConaghy Trent. 2011. FFX: Fast, scalable, deterministic symbolic regression technology. In Genetic Programming Theory and Practice IX. Springer, 235–260. http://link.springer.com/chapter/10.1007/978-1-4614-1770-5_13

[26]. Medernach David, Fitzgerald Jeannie, Azad R. Muhammad Atif, and Ryan Conor. 2016. A New Wave: A Dynamic Approach to Genetic Programming. In Proceedings of the Genetic and Evolutionary Computation Conference 2016 (GECCO '16). ACM, New York, NY, USA, 757–764. 10.1145/2908812.2908857

[27]. Muharram Mohammed and Smith George D. 2005. Evolutionary constructive induction. IEEE Transactions on Knowledge and Data Engineering 17, 11 (2005), 1518–1528. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1512037

[28]. Muñoz Luis, Silva Sara, and Trujillo Leonardo. 2015. M3GP–Multiclass Classification with GP. In Genetic Programming. Springer, 78–91. http://link.springer.com/chapter/10.1007/978-3-319-16501-1_7

[29]. Muñoz Luis, Trujillo Leonardo, Silva Sara, Castelli Mauro, and Vanneschi Leonardo. 2018. Evolving multidimensional transformations for symbolic regression with M3GP. Memetic Computing (Sept. 2018). 10.1007/s12293-018-0274-5

[30]. Neshatian Kourosh, Zhang Mengjie, and Andreae Peter. 2012. A filter approach to multiple feature construction for symbolic learning classifiers using genetic programming. IEEE Transactions on Evolutionary Computation 16, 5 (2012), 645–661.

[31]. Olson Randal S., Cava William La, Orzechowski Patryk, Urbanowicz Ryan J., and Moore Jason H. 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.

[32]. Orzechowski Patryk, Cava William La, and Moore Jason H. 2018. Where are we now? A large benchmark study of recent symbolic regression methods. arXiv:1804.09331 [cs] (April 2018). 10.1145/3205455.3205539 arXiv: 1804.09331.

[33]. Pawlak Tomasz P., Wieloch Bartosz, and Krawiec Krzysztof. 2015. Semantic backpropagation for designing search operators in genetic programming. IEEE Transactions on Evolutionary Computation 19, 3 (2015), 326–340.

[34]. Pedregosa Fabian, Varoquaux Gaël, Gramfort Alexandre, Michel Vincent, Thirion Bertrand, Grisel Olivier, Blondel Mathieu, Prettenhofer Peter, Weiss Ron, Dubourg Vincent, and others.

2011. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12, Oct (2011), 2825–2830. http://www.jmlr.org/papers/v12/pedregosa11a.html

[35]. Schmidt Michael and Lipson Hod. 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

[36]. Silva Sara, Munoz Luis, Trujillo Leonardo, Ingalalli Vijay, Castelli Mauro, and Vanneschi Leonardo. 2015. Multiclass Classification Through Multidimensional Clustering. In Genetic Programming Theory and Practice XIII. Vol. 13. Springer, Ann Arbor, MI.

[37]. Tibshirani Robert. 1996. Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society. Series B (Methodological) (1996), 267–288. http://www.jstor.org/stable/2346178

[38]. Tibshirani Robert, Hastie Trevor, Narasimhan Balasubramanian, and Chu Gilbert. 2002. Diagnosis of multiple cancer types by shrunken centroids of gene expression. Proceedings of the National Academy of Sciences 99, 10 (May 2002), 6567–6572. 10.1073/pnas.082099299

[39]. Vanneschi Leonardo, Castelli Mauro, Manzoni Luca, Krawiec Krzysztof, Moraglio Alberto, Silva Sara, and Gonçalves Ivo. 2017. PSXO: population-wide semantic crossover. In Proceedings of the Genetic and Evolutionary Computation Conference Companion. ACM, 257–258.

[40]. de Melo Vinícius Veloso and Banzhaf Wolfgang. 2017. Automatic Feature Engineering for Regression Models with Machine Learning: an Evolutionary Computation and Statistics Hybrid. Information Sciences (2017). 10.1016/j.ins.2017.11.041

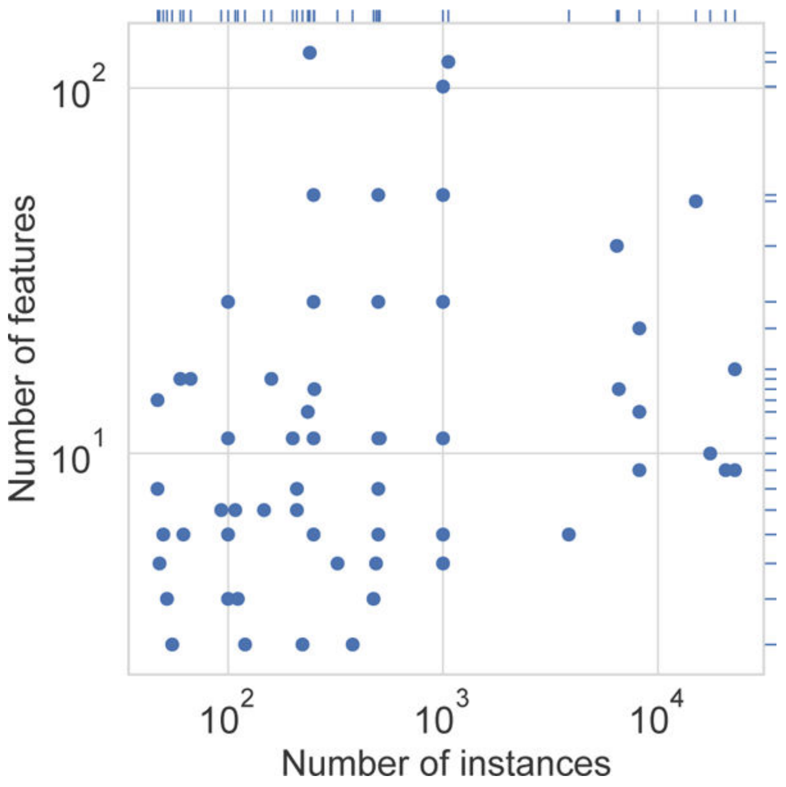

Figure 1:
Properties of the benchmarks used from PMLB [31].

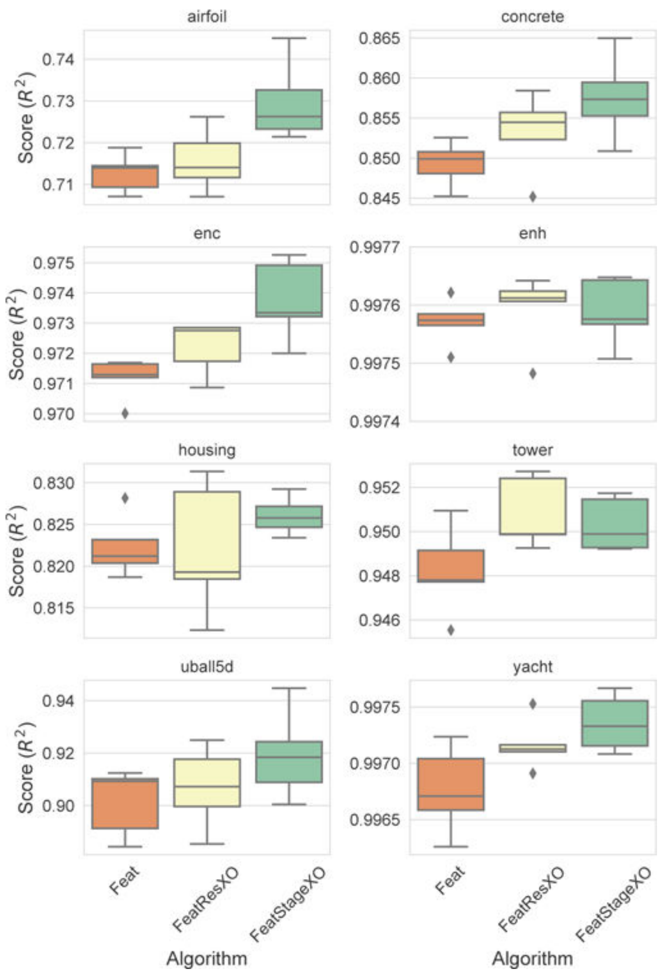

Figure 2:
Mean 5-fold CV R^2 performance for different crossover operators on the 8 tuning problems.

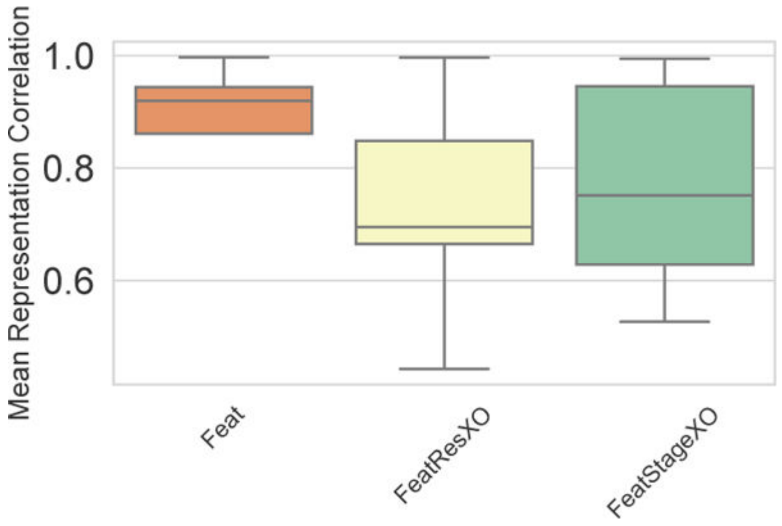

Figure 3:
Average pairwise representation correlation (Eqn. 3) for different crossover operators on the 8 tuning problems.

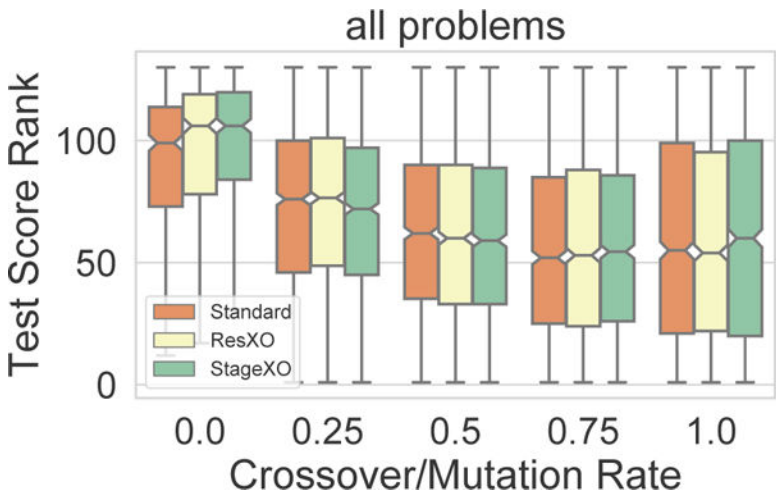

Figure 4:
Test rankings for different crossover probabilities on the 8 tuning problems.

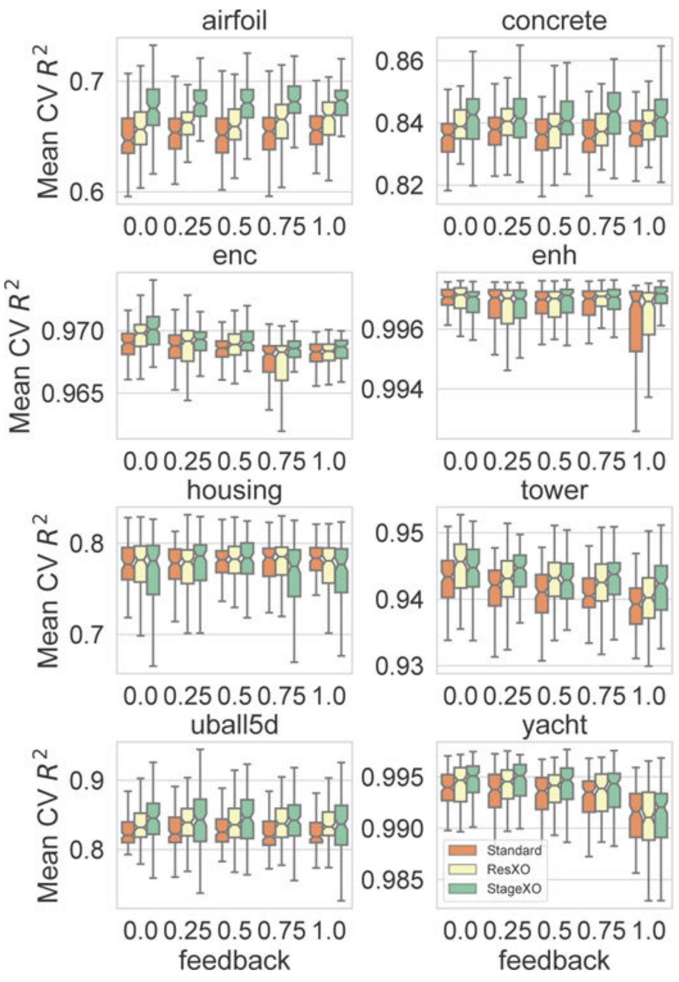

Figure 5:

Mean 5-fold CV R^2 performance for different levels of feedback on the 8 tuning problems.

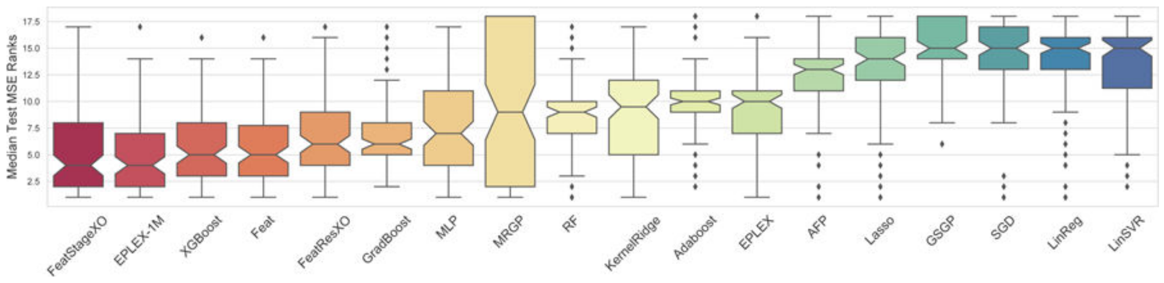

Figure 6:
Median test MSE rankings on the PMLB datasets. The box shows the quartiles of the rankings with whiskers showing the rest of the distribution excluding outliers. Algorithms are ordered left to right by best (lowest) to worst (highest) median ranking.

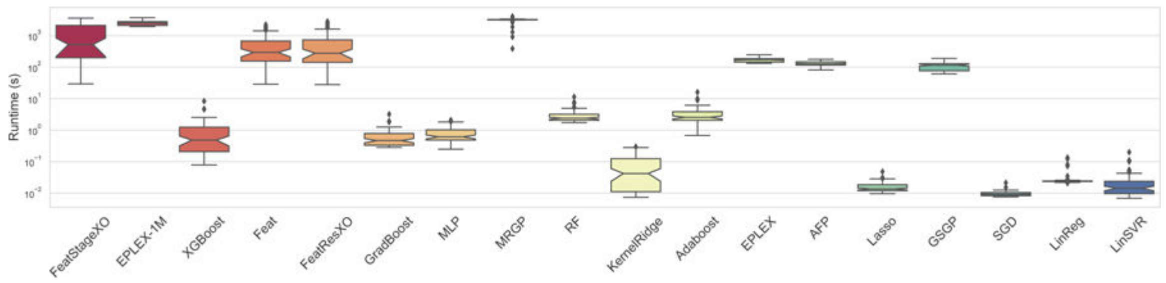

Figure 7:
Runtime comparisons on the PMLB datasets. Runtime is the wall clock time for a single training instance. Algorithms are ordered to match Fig. 6.

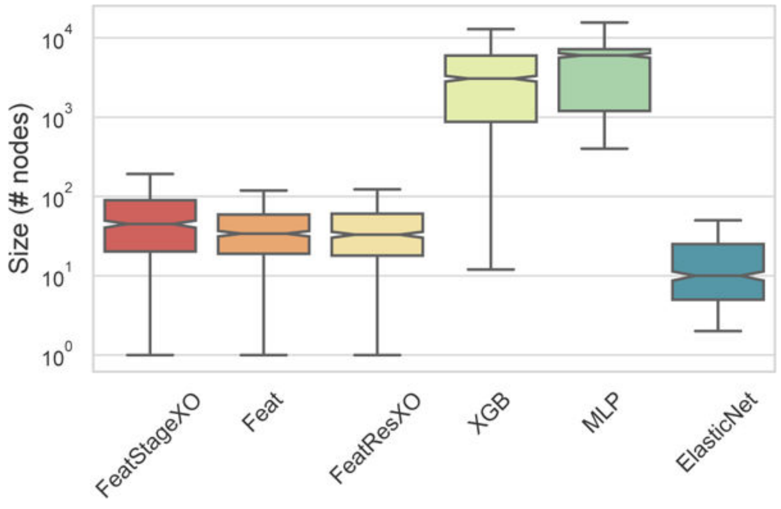

Figure 8:
Number of nodes in solutions on the PMLB datasets.

Table 1:

Hyperparameter values for FEAT in the experiments. The bottom values are fixed.

Hyperparameter	**Values**
probability of crossover (complement: mutation)	[0,0.25,0.5, 0.75,1.0]
feedback (γ, Eqn. 2)	[0, 0.25, 0.5, 0.75, 1.0]
type of feature crossover	[Standard, ResXO, StageXO]
probability of feature crossover (complement: subtree crossover)	[0.5, 0.75, 1.0]
feedback softmax normalization	[On, Off]
population size	500
generations	100 (200 for PMLB)
max depth	6
maximum dimensionality	min(50, 2 * |x|)
iterations of gradient descent	10

Table 2:

Regression problems used for method comparisons.

Problem	Dimension	Samples
Airfoil	5	1503
Concrete	8	1030
ENC	8	768
ENH	8	768
Housing	14	506
Tower	25	3135
UBall5D	5	6024
Yacht	6	309

Table 3:

Bonferroni-adjusted *p*-values using a Wilcoxon signed rank test of R^2 scores for the methods across all tunng problems. Bold: $p < 0.05$.

	Feat	**FeatResXO**
FeatResXO	4.6e-01	
FeatStageXO	**3.5e-02**	1.2e-01

Table 4:

Best hyperparameter values for FEAT across the 8 tuning problems.

Hyperparameter	**FEAT**	**FEAT-ResXO FEA**	**T-StageXO**
probability of crossover	0.75	0.75	0.75
feedback	0.25	0.0	0.25
probability of feature crossover	0.75	0.5	0.5
feedback softmax normalization	Off	Off	Off

Table 5:

Bonferroni-adjusted *p*-values using a Wilcoxon signed rank test of MSE scores for the methods across all benchmarks. Bold: p <0.05.

	Adaboost	AFP	EPLEX	EPLEX-1M	FEAT	FEATResXO	FEATStageXO	GradBoost	GSGP	KernelRidge	Lasso	LinReg	LinSVR	MLP	MRGP	RF	SGD
AFP	**1.2e-02**																
EPLEX	1.0e+00	**7.3e-08**															
EPLEX-1M	**8.2e-09**	**9.4e-12**	**3.4e-09**														
FEAT	**2.6e-08**	**5.8e-10**	**1.9e-06**	1.0e+00													
FEATResXO	**1.8e-04**	**2.2e-06**	**5.8e-04**	**5.0e-02**	**1.7e-02**												
FEATStageXO	**9.1e-08**	**8.2e-10**	**4.1e-06**	1.0e+00	1.0e+00	**5.8e-03**											
GradBoost	**1.3e-08**	**2.3e-08**	**5.6e-03**	**3.7e-02**	1.0e+00	1.0e+00	1.0e+00										
GSGP	**8.4e-11**	**2.6e-07**	**4.1e-12**	**1.5e-14**	**8.1e-14**	**5.1e-13**	**2.3e-14**	**2.6e-14**									
KernelRidge	1.0e+00	**1.5e-03**	1.0e+00	**8.1e-04**	**2.1e-02**	1.0e+00	**1.1e-02**	9.9e-01	**1.4e-14**								
Lasso	**8.3e-04**	1.0e+00	**1.9e-06**	**4.4e-12**	**4.5e-10**	**3.5e-07**	**4.8e-10**	**1.8e-08**	**2.1e-02**	**8.6e-08**							
LinReg	**1.1e-04**	**7.0e-03**	**2.3e-07**	**3.7e-12**	**4.5e-11**	**4.5e-09**	**6.1e-11**	**1.7e-08**	1.0e+00	**3.0e-09**	1.0e+00						
LinSVR	**1.1e-03**	8.6e-02	**2.3e-06**	**4.6e-12**	**1.3e-09**	**5.7e-07**	**9.0e-10**	**2.6e-08**	3.9e-01	**7.9e-09**	1.0e+00	1.0e+00					
MLP	**2.6e-02**	**2.6e-06**	1.0e+00	1.0e+00	1.0e+00	1.0e+00	1.0e+00	1.0e+00	**9.6e-15**	7.4e-01	**6.8e-07**	**2.0e-08**	**1.3e-07**				
MRGP	1.0e+00	1.0e+00	1.0e+00	**1.1e-04**	**7.0e-04**	2.8e-01	**3.6e-04**	2.1e-01	**2.1e-05**	1.0e+00	2.3e-01	**2.5e-02**	1.5e-01	1.0e+00			
RF	**7.9e-05**	**1.2e-04**	1.0e+00	**7.4e-06**	**1.0e-04**	2.0e-01	**7.6e-05**	**2.3e-06**	**2.1e-13**	1.0e+00	**4.7e-06**	**4.4e-06**	**1.3e-06**	1.0e+00	1.0e+00		
SGD	**2.0e-06**	**6.4e-05**	**1.9e-09**	**9.6e-13**	**2.7e-12**	**1.5e-09**	**1.3e-11**	**4.1e-11**	1.0e+00	**2.6e-09**	5.5e-02	1.0e+00	1.0e+00	**1.8e-09**	**4.5e-03**	**2.8e-09**	
XGBoost	**2.1e-08**	**6.9e-11**	**5.3e-05**	1.0e+00	1.0e+00	1.0e+00	1.0e+00	**8.9e-03**	**6.0e-15**	**6.9e-03**	**1.3e-10**	**6.4e-10**	**2.2e-10**	1.0e+00	**2.8e-03**	**3.6e-08**	**5.8e-12**