A General Feature Engineering Wrapper for Machine Learning Using ϵ-Lexicase Survival

William La Cava(✉) and Jason Moore

Institute for Biomedical Informatics, University of Pennsylvania,
Philadelphia, PA 19104, USA
{lacava,jhmoore}@upenn.edu

Abstract. We propose a general wrapper for feature learning that interfaces with other machine learning methods to compose effective data representations. The proposed feature engineering wrapper (FEW) uses genetic programming to represent and evolve individual features tailored to the machine learning method with which it is paired. In order to maintain feature diversity, ϵ-lexicase survival is introduced, a method based on ϵ-lexicase selection. This survival method preserves semantically unique individuals in the population based on their ability to solve difficult subsets of training cases, thereby yielding a population of uncorrelated features. We demonstrate FEW with five different off-the-shelf machine learning methods and test it on a set of real-world and synthetic regression problems with dimensions varying across three orders of magnitude. The results show that FEW is able to improve model test predictions across problems for several ML methods. We discuss and test the scalability of FEW in comparison to other feature composition strategies, most notably polynomial feature expansion.

Keywords: Genetic programming · Feature selection · Representation learning · Regression

1 Introduction

The success of machine learning (ML) algorithms in generating predictive models depends completely on the representation of the data used to train them. For this reason, and given the accessibility of many standard ML implementations to today's researchers, feature selection and synthesis are quickly becoming the most pressing issues in machine learning. The goal of feature engineering, i.e. feature learning or representation learning [2], is to learn a transformation of the attributes that improves the predictions made by a learning algorithm. Formally, given N paired examples of d attributes from the training set $\mathcal{T} = \{(\mathbf{x}_i, y_i), i = 1 \ \ldots \ N\}$, we wish to find a P-dimensional feature mapping $\Phi(\mathbf{x}) : \mathbb{R}^d \rightarrow \mathbb{R}^P$ for a regression model $\hat{y}(\Phi(\mathbf{x})) : \mathbb{R}^P \rightarrow \mathbb{R}$ that performs better than the model $\hat{y}(\mathbf{x}) : \mathbb{R}^d \rightarrow \mathbb{R}$ formed directly from $\mathbf{x}$. Doing so is a significant challenge, since it is not straightforward to determine useful nonlinear transformations of the raw feature set that may prove consequential in predicting

J. McDermott et al. (Eds.): EuroGP 2017, LNCS 10196, pp. 80–95, 2017.
DOI: 10.1007/978-3-319-55696-3_6

the phenomenon in question. It is known that optimal feature selection (that is, selecting the optimal subset of features from a dataset) is itself hard [4], not to mention the process of determining an optimal feature representation. Although approaches to feature expansion are well known, for example polynomial basis expansion, these methods must be paired with a feature selection or parameter regularization scheme to control model complexity [5]. As we argue and show in this paper, feature engineering using genetic programming (GP) can be competitive in this task of optimizing the data representation for several ML methods, leading to more predictive regression models with a controlled number of features.

Typically GP is applied to regression by constructing and optimizing a population of symbolic models. Symbolic regression is a useful method for generating meaningful model structures for real-world applications [13] due to its flexibility in model representation. This comes at the expense of high computational cost, due to the expansive search space required to search model structures and parameters. In comparison, many ML approaches only attempt optimize the parameters of a single, fixed model structure with respect to a loss function, such as the mean squared error (linear regression) or the ϵ-insensitive loss (support vector regression). Recent studies in GP [1,3] suggest that computation time can be saved by narrowing the scope of search to the space of model building-blocks rather than models themselves, and then generating a model via linear regression over the population outputs. Such an approach presents new challenges, since the population should no longer be pressured to converge on a single model, but rather to reach a set of transformations that are more or less orthogonal to each other. In this paper, we extend previous work by introducing a general framework for feature engineering that is agnostic with respect to the ML method used, and that modifies a recent parent selection technique [14] to maintain an uncorrelated population of features. The proposed feature engineering wrapper (FEW) is a GP-based method that interfaces with Scikit-learn [21] to provide generic feature engineering for its entire suite of learners. It is available open-source[1] as a Python package via the Python Package Index (PyPi)[2].

In this paper we demonstrate FEW in conjunction with Lasso [25], linear and non-linear support vector regression (SVR), K-Nearest Neighbors (KNN) regression, and decision tree (DT) regression. We show that in many cases, FEW is able to significantly improve the learners with which it is paired. We also demonstrate that for problems with large numbers of attributes, FEW is able to effectively search the feature space to generate models with a smaller set of optimized features, a task that is infeasible for some brute-force feature expansion methods. Central to the ability of FEW to learn a set of features rather than a single redundant feature is its use of a new survival method, known as ϵ-lexicase survival, which is introduced in Sect. 2.1. ϵ-lexicase suvival is a variant of ϵ-lexicase selection [14] that maintains a population of individuals with unique outputs and

[1] https://github.com/lacava/few.
[2] https://pypi.python.org/pypi/FEW.

that survive based on their performance in the context of both the population semantics and the difficulty of the training cases.

We describe FEW in detail in Sect. 2, including a description of the survival algorithm as well as a discussion the scalability of this approach compared to other feature composition strategies. We review related work in Sect. 3, both in GP and in representation learning. The experiments are described in Sect. 4 and include a hyper-parameter optimization study of FEW and an analysis of FEW on a number of real-world problems and ML algorithms in comparison to polynomial basis expansion. The results are presented in Sect. 5 in terms of model accuracy on test sets and the rankings of methods across problems. We discuss the results and future work in Sect. 6 and conclude in Sect. 7.

2 Feature Engineering Wrapper

FEW uses GP optimize a population of feature transformations that are used to generate a predictive model each generation using a user-defined ML method. It continues to optimize the feature transformations as a GP population while building a single predictive model at the beginning of each subsequent generation. The steps of FEW are shown in Fig. 1.

FEW begins by fitting a model using a given ML method and the original attributes. This model is stored, along with its score on an internal validation set, and updated whenever a model is found with a better cross-validation (CV) score. This guarantees that the model generated by FEW performs at least as well on the internal validation set as the underlying ML method with which it is paired. FEW then constructs an initially random population of engineered features represented as $\Phi^k(\mathbf{x})$ in Fig. 1. This initial population is seeded with any original attributes with non-zero coefficients (for ML methods that use coefficients); for example, Φ^k may equal $[x_1,\ x_2,\ \phi_3,\ \ldots,\ \phi_P]$, where x_1 and x_2 are seeded from the initial ML model, and $\phi_3\ \ldots\phi_p$ are initialized randomly. FEW then loops through the processes of selection, variation, fitness, survival, and ML model construction.

The selection phase is treated as an opportunity for feedback from the ML model on the current population of features, if any, to bias the parents chosen for variation. For example, some learners (e.g. Lasso, SVR) implement ℓ_1 regularization to minimize feature coefficients as a form of feature selection. This information is incorporated during selection by removing individuals with coefficients of zero. Otherwise, selection for variation is random; note that selection is designed to be generally weak in order to prevent population convergence. The selected population is represented by $\Phi^{k'}$ in Fig. 1.

Following selection, the individuals in the population are varied using subtree crossover and point mutation to produce individuals $\Phi^{\text{offspring}}$. The fitness of each feature is then assessed. It is tempting to tailor the fitness function to each individual ML algorithm, but in the interest of time and scope, fairly conventional fitness metrics are used here. For the ML methods designed to minimize mean squared error (MSE), the MSE metric is used in fitness assessment. For

SVR, where the ϵ-insensitive loss function is minimized, the mean absolute error (MAE) is used.

After fitness assessment, the set consisting of the parents and offspring ($\Phi^{k'}$ and $\Phi^{\text{offspring}}$) compete for survival, resulting in a reduction of the population back to its original size. This updated population of features, Φ^{k+1} in Fig. 1, is then used to fit a new ML model and the process repeats.

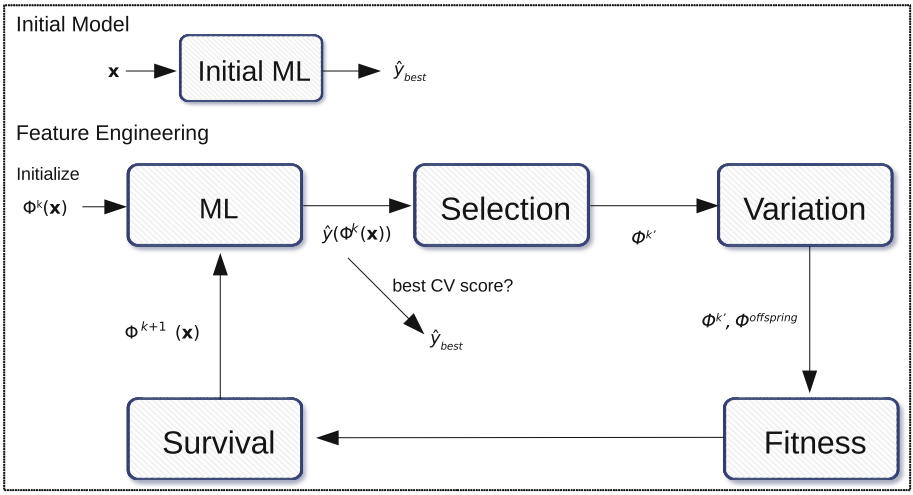

Fig. 1. Steps in the FEW process. After fitting an initial ML model, a set of feature transformations are initialized in a GP population. These engineered features are used to fit a new ML model each generation. The engineered features are then updated via selection, variation, fitness assessment and survival. The final model returned is that with the highest internal validation score among models generated.

Defining a GP system in such a way presents unique challenges. The individuals in the population not only have to compete to be effective features to solve the problem, but must work well in combination with the rest of the population. As a result, it is undesirable to have redundant i.e. correlated features, or to have the population converge to a single behavior. The goal instead is to have a population of features that essentially minimize the residuals of the outputs of other individuals in the population. As such, we are motivated to use ϵ-lexicase selection [14] as the survival scheme for individuals, since this method rewards individuals for performing well on unique subsets of the training samples. This method shifts selection pressure continually to training cases that have are most difficult for the population to solve, as defined by population performance. It also has been shown to maintain very high semantic diversity in the population [8,14], which should result in un-correlated engineered features.

2.1 ϵ-lexicase Survival

Ideally, the population would consist of uncorrelated features that complement each other in the construction of a single model. Central to this goal is maintaining diversity among individual outputs in the population while continuing to improve model prediction. With this in mind, we propose a new method for the survival routine called ϵ-lexicase survival that chooses individuals for survival based on their performance on unique subsets of training cases. It implements a slight alternation of ϵ-lexicase selection [14], a recent method for parent selection based on lexicase selection [8,23], in order to (1) make ϵ-lexicase return unique individuals and (2) make ϵ-lexicase work for survival rather than selection. The basic ϵ-lexicase survival strategy is as follows:

1. `Initialize`
   (a) Set **candidates** to be the remaining (unselected) population of programs.
   (b) Set **cases** to be a list of all of the training cases in the training set in random order.
2. `Loop`
   (a) Set **candidates** to be the subset of the current **candidates** that are within ϵ of the best performance of any individual currently in **candidates** for the first case in **cases**.
   (b) If **candidates** contains just a single individual then return it.
   (c) If **cases** contains just a single training case then return a randomly selected individual from **candidates**.
   (d) Otherwise remove the first case from **cases** and go to `Loop`.

This routine is repeated until the surviving population matches the user-defined population size. Several methods of determining ϵ were presented in the original paper [14]; we use the most successful one, which defines ϵ based on the median absolute deviation of fitnesses on the training case in question. In other words, for each training case i, ϵ_i is defined as

$$\epsilon_i = \text{median}\left(|\mathbf{e}_i - \text{median}(\mathbf{e}_i)|\right) \tag{1}$$

where $\mathbf{e}_i \in \mathbb{R}^{|2P|}$ is the fitness on training case i of every individual in the set consisting of parents and their offspring. This definition of ϵ is nice because it is parameter free, adjusts to accommodate differences in hardness of training cases and population performance, and has been demonstrated to perform better than user-defined versions [13]. Two other differences between ϵ-lexicase *survival* and *selection* should be noted. First, unlike ϵ-lexicase selection, once an individual has been chosen for survival, it is removed from the survival pool in step 1.(a). This is to prevent the same individual from being selected multiple times, which would lead to redundant features in the population. In addition, 2.(a) in ϵ-lexicase survival is more similar to lexicase selection [23] in that the best error is defined relative to the current candidates rather than the whole population. In ϵ-lexicase selection, this was not the case.

2.2 Scaling

It is worth considering the complexity of the proposed method in the context of other feature engineering methods, namely basis function expansions and kernel transformations. Basis function expansions may scale exponentially; for example a n-degree polynomial expansion of d features grows as $O(d^n)$. Kernel representations, meanwhile, scale with the number of samples; the complexity of computing a kernel is $O(N^2)$ and computing the solution to a kernelized regression problem with parameter regularization is $O(N^3)$ [5]. In comparison, FEW scales linearly with N, and otherwise scales according to the population size P and the number of generations g set by the user. This scaling can be advantageous compared to basis function expansion when the number of features is large, since the user controls the number of features to be included in the ML model (P), as well as the training time via g. For large sample sizes, FEW may be advantageous with respect to kernel expansion as well. The survival method used determines how FEW scales with respect to P. ϵ-lexicase survival has a worst-case complexity of $O(P^2N)$ in selecting a suvivor, compared to $O(PN)$ for tournament selection. In practice, however, it should be noted that no significant difference in run-time has been observed between tournament and ϵ-lexicase selection [14], due to the fact that ϵ-lexicase selection normally only uses a small set of training examples for each selection event. Although FEW can be expected to scale independently of the number of attributes, it should be noted that larger population sizes or more generations may be necessary to achieve satisfactory results for larger data sets.

3 Related Work

Hybrid learning methods that incorporate constant optimization in GP are quite common [9,12,26]. Others have proposed two-step approaches designed to first generate a set of features using GP and then construct a deterministic ML model. For example, evolutionary constructive induction methods [17–19] have been proposed to learn features for decision trees. Another method, FFX [16], applies a non-evolutionary, two-step technique that generates randomized basis functions for linear regression. FFX was coupled with a third symbolic regression stage in later research [10]. FEW's wrapper-based approach is inspired by evolutionary feature synthesis (EFS) [1], an algorithm that iterates upon a population of synthesized features while using them to fit a Lasso model. EFS is perhaps better classified as a population-based stochastic hill climbing method since it does not incorporate typical crossover or selection stages. Nevertheless it demonstrated the ability of feature space to be searched with EC-inspired search operators. FEW differs from EFS in its reliance on ϵ-lexicase survival to maintain population diversity, and its use of crossover as a search operator. It also opts for a single population size parameter to minimize the number of hyper-parameters. Unlike the methods mentioned above, FEW can be paired with any ML estimator in Scikit-learn [21], whereas previous approaches have been tailored to use a fixed ML method (linear regression, Lasso or DT).

In the context of non-evolutionary methods, other feature engineering/selection techniques are common, such as basis function expansions, feature subset selection via forward and backward selection [6], and regularization. Typically the feature space is augmented via the basis function expansion, and then either a feature selection method is used to select a subset of features prior to learning, or regularization pressure is applied to perform implicit feature selection during learning by minimizing the coefficients of some features to zero [5]. This approach is of course quite different from what is proposed in FEW: first, basis expansion relies on enumeration of the expanded set features (unless the kernel trick is used, in which case interpretability is lost); second, it either relies on a greedy approach to solving the feature selection problem [4] or relies on the regularization penalty to provide an acceptable level of sparseness in the resultant coefficients, which is not guaranteed.

In the context of meta-learning, FEW can be viewed as a wrapper concerned specifically with feature construction for a given estimator rather than a hyper-optimization strategy such as TPOT [20], which optimizes machine learning pipelines, or HyperOpt [11], which optimizes parameters of ML pipelines. It would be straightforward to include FEW as an estimator in either of these systems.

4 Experimental Analysis

A set of 7 real-world and 1 synthetic problems were used to analyze FEW, ranging in sample size (506–21048) and number of features (5–529). As a first step, we conducted a hyper-parameter optimization experiment, varying the population size and survival method for FEW. This hyper-parameter tuning step is conducted on three of the studied problems with Lasso set as the ML method. To make the population size somewhat normalized to the tested problem, we specified population size as a fraction of the problem dimensions; e.g. 0.5× indicates a population size equal to half the number of raw features and 4x indicates a population size equal to four times the number of raw features. In the second step, each ML method was tested with and without FEW, and compared in terms of the accuracy of the generated models on the test set as measured by the coefficient of determination:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i \in \mathcal{T}} (y_i - \hat{y}_i)^2}{\sum_{i \in \mathcal{T}} (y_i - \bar{y})^2}$$

where $\bar{y}$ is the mean of the target output y. Note that R^2 in this case can be negative, since the variance of the residual of the model prediction can exceed the variance in the data. The FEW settings and problem descriptions are summarized in Table 1.

Each paired ML method uses the default settings from Scikit-learn. For Lasso, the `LassoLarsCV()` method is used, which is a least-angle regression implementation of Lasso that uses internal cross-validation to tune the regularization

parameter. Support vector regression defaults to using a radial basis function kernel. KNN defaults to $k = 5$. The following section describes the set of problems used.

Table 1. FEW problem settings. The bold values indicate those chosen for the main analysis based on hyper-parameter validation.

Setting	Value		
Population size	**0.5x**, 1x, 2x, **3x**, 4x		
Survival method	tournament, **ϵ-lexicase**		
Program depth limit	2		
Generations	100		
Crossover/mutation	50/50%		
Elitism	keep best		
Trials	30 shuffled 70/30% splits		
Machine learner	Lasso, Linear SVR, KNN, SVR, DT		
Terminal set	$\{\mathbf{x}, +, -, *, /, \sin(\cdot), \cos(\cdot), \exp(\cdot), \log(\lvert\cdot\rvert), (\cdot)^2, (\cdot)^3\}$		
Problem	Dimension	Training samples	Test samples
UBall5D	5	1024	5000
ENH	8	538	230
ENC	8	538	230
Housing	14	354	152
Tower	25	2195	940
Crime	128	1395	599
UJI_long	529	14734	6314
UJI_lat	529	14734	6314

4.1 Problems

The UBall5D problem, also known as Vladislavleva-4, is the one synthetic problem studied whose solution is of the form $y = \frac{10}{5+\sum_{i=1}^{5}(x_i-3)^2}$, which is known to be difficult to solve exactly [30]. The second and third problem tasks are to estimate the energy efficiency of heating (ENH) and cooling (ENC) requirements for various simulated buildings [28]. The housing data set [7] seeks a model to estimate Boston housing prices. The Tower problem consists of 15-minute averaged time series data taken from a chemical distillation tower, with the goal of predicting propelyne concentration. The Tower problem and UBall5D were chosen from the benchmark suite suggested by White et al. [31].

FEW is tested on three problems with large numbers of features. For these problems a population size of 0.5× is used in order to achieve dimensionality reduction in the model. The Crime problem consists of 128 community features used to estimate the per capita violent crime rates across the United States [22].

The UJI_long and UJI_lat problems are benchmark data sets for predicting the latitude and longitude, respectively, of indoor users from WLAN signals [27]. The dimensions of all data sets are shown in Table 1. The data sets were divided 70/30 into training and testing sets for 30 trials.

5 Results

The results of the hyper-parameter optimization are presented first, for which the FEW settings for the main set of problem comparisons are chosen. Afterwords we present a comparison of model fitness using the raw features, using polynomial feature expansion, and using FEW for each of the ML methods.

5.1 Hyper-Parameter Optimization

The test R^2 values of the models generated using each combination of hyper-parameters are shown in boxplot form in Fig. 2. As the population size is increased, and therefore the number of features presented to the ML model, the models produced tend to perform better, as expected. This performance improvement appears to level off for the Housing and Tower problem with a population size of 3×. For this reason we choose 3× as the population size in our subsequent experiments. Also note that ϵ-lexicase survival outperforms tournament survival for larger population sizes, including 3×. This observation complements other experiments with lexicase selection that show its performance to be dependent on the number of training cases [15]. ϵ-lexicase's good performance may be explained in part by the increase in output diversity among the engineered features, shown in Fig. 3. Here diversity is measured as one minus the average feature correlations, such that a higher value indicates less correlated features.

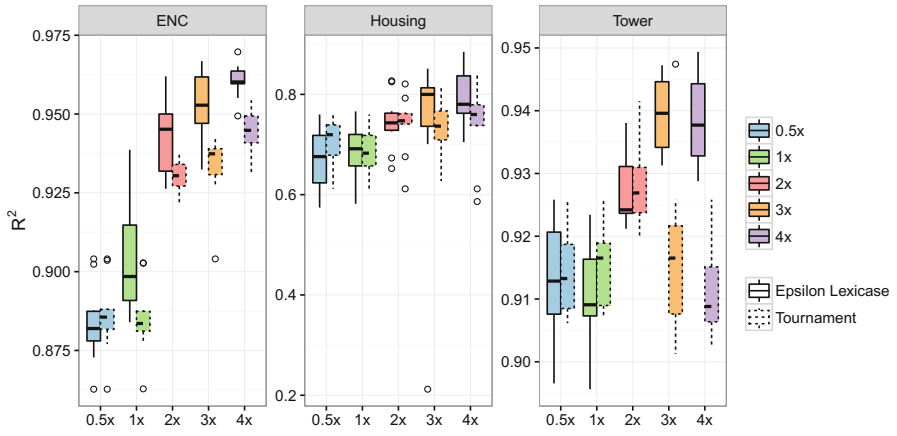

Fig. 2. Comparison of test set accuracy for various population sizes and selection methods.

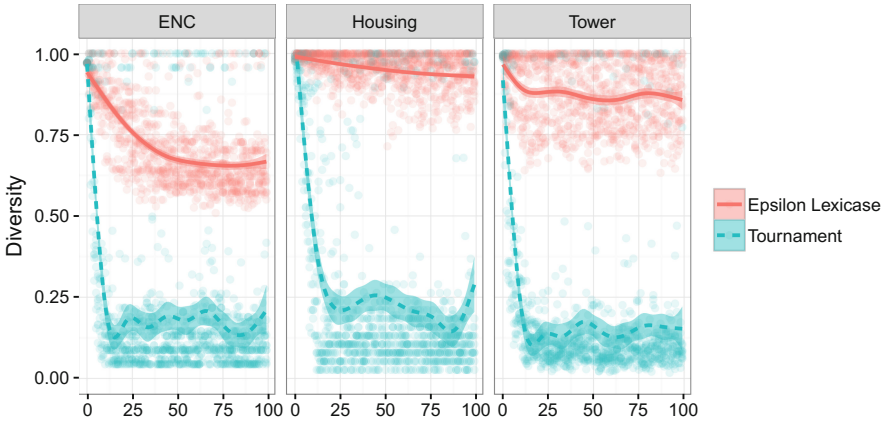

Fig. 3. Diversity of engineered features using ϵ-lexicase survival or tournament survival with 3× population size.

5.2 Problem Performance

The test scores on each problem are shown in Fig. 4 with respect to the ML algorithm pairing (Lasso, LinearSVR, KNN, SVR, DT) and the feature set used (raw, 2-degree polynomial, FEW). The final subplot in Fig. 4 shows the ranking statistics for each method across all problems. It is important to note that the polynomial feature expansion ran out of memory for the UJIIndoor problems, and hence no results for that method are included for those two problems. The memory error for this problem is reasonable given that a 2-degree expansion of 529 features would result in more than 279,000 features.

With the exception of its use with Lasso, FEW outperforms polynomial feature expansion for each ML method in terms of rankings across problems. The polynomial expansion worsens the ML rankings for KNN, SVR and DT compared to using the raw features, suggesting that these methods overfit with polynomial features. Conversely, FEW improves the model rankings across problems for all ML methods except for DT, on which it performs roughly the same as the raw features. Over all problems and methods, SVR paired with FEW ranks the best, followed by KNN paired with FEW.

5.3 Statistical Analysis

We conduct pairwise Wilcoxon rank-sum tests to compare the feature representation methods for each ML method and problem, correcting for multiple comparisons. The detailed results are presented in Table 2. In comparison to using the raw features, FEW works best in conjunction with linear and nonlinear SVR, in which cases it significantly improves model accuracy for 6 out of 8 problems. FEW improves Lasso on 5 problems, KNN on 3 problems, and

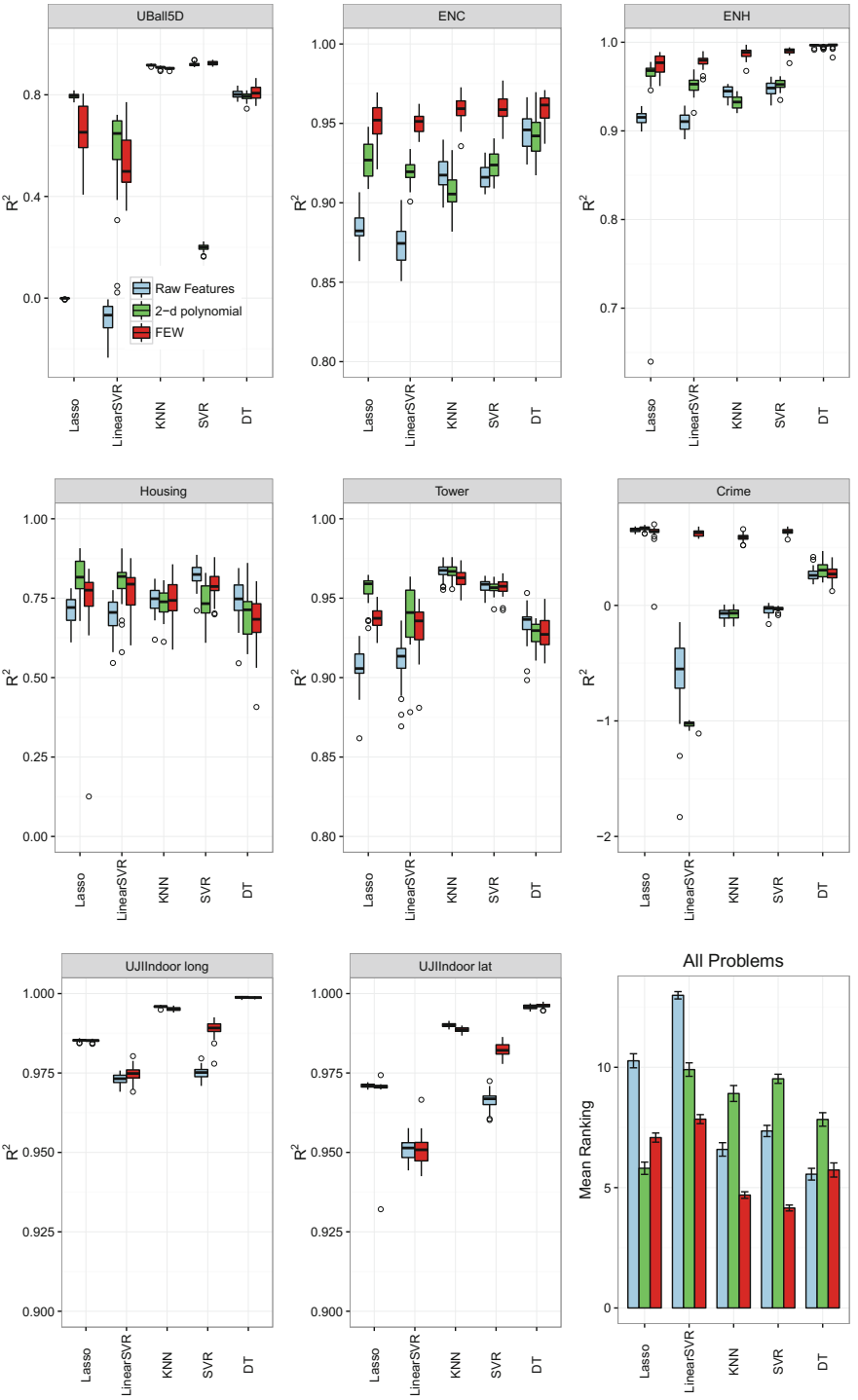

Fig. 4. Comparison of results for ML methods and FEW-enabled methods.

Table 2. p-values based on a Wilcoxon rank-sum test with Holm correction for multiple comparisons. Pairwise comparisons of the feature methods (FEW, Raw, and 2d poly) are grouped by the ML method and problem. Yellow highlighting indicates that the first feature method is significantly better than the second, whereas gray highlighting indicates that the first feature method is significantly worse than the second ($p < 0.05$).

	UBall5D	ENC	ENH	Housing	Tower	Crime	UJIIndoor Long	UJIIndoor Lat
Lasso								
FEW - Raw	0.0000	0.0000	0.0000	0.0004	0.0000	0.0287	0.0714	0.0208
2d poly - Raw	0.0000	0.0000	0.0000	0.0000	0.0000	0.0105	N/A	N/A
FEW- 2d poly	0.0000	0.0000	0.0024	0.0020	0.0000	0.0001	N/A	N/A
Linear SVR								
FEW - Raw	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0564	0.0346
2d poly - Raw	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	N/A	N/A
FEW- 2d poly	0.1646	0.0000	0.0000	0.0428	0.0584	0.0000	N/A	N/A
KNN								
FEW - Raw	0.0000	0.0000	0.0000	0.4471	0.0004	0.0000	0.0000	0.0000
2d poly - Raw	0.0000	0.0005	0.0000	0.3219	0.6152	0.9528	N/A	N/A
FEW- 2d poly	0.0002	0.0000	0.0000	0.1473	0.0016	0.0000	N/A	N/A
SVR								
FEW - Raw	0.0149	0.0000	0.0000	0.0025	0.2973	0.0000	0.0000	0.0000
2d poly - Raw	0.0000	0.0071	0.0321	0.0000	0.2369	0.7901	N/A	N/A
FEW- 2d poly	0.0000	0.0000	0.0000	0.0006	0.1563	0.0000	N/A	N/A
DT								
FEW - Raw	0.5444	0.0000	0.1646	0.0005	0.0114	0.2625	0.1331	0.0073
2d poly - Raw	0.0230	0.1359	0.0011	0.0048	0.0002	0.0399	N/A	N/A
FEW- 2d poly	0.0148	0.0000	0.0005	0.4688	0.9882	0.0644	N/A	N/A

DT on 2 problems. The effects of FEW on KNN and DT are mixed, since FEW performs worse than the raw features for 4 and 2 problems, respectively. In comparison to 2d polynomial feature expansion, FEW is less effective when paired with Lasso, performing better in 2 problems and worse for 4 problems out of 6, all of which are statistically significant. However, when paired with linear and nonlinear SVR, FEW performs better than polynomial features on 3 and 5 problems, respectively. FEW also performs significantly better than polynomial features when paired with KNN, and DT, in terms of numbers of problems.

We also conduct an analysis of variance (ANOVA) test by sampling the mean rankings of each ML and feature method for each trial over all problems (i.e. for the data corresponding to the bottom right plot in Fig. 4). In this case the UJIIndoor results are excluded since the polynomial method failed to produce models. The ANOVA suggests significant differences in the ML methods ($p = 0.02117$) and feature representations ($p = 0.0015$). A post-hoc test (Tukey's multiple comparisons of means) is conducted to determine the source of differences in feature representations; the results are shown in Table 3. The test shows

Table 3. Tukey multiple comparisons of means, 95% family-wise confidence level. Test of significance for feature engineering rankings across problems and methods.

Comparison	difference	lower	upper	adjusted p-value
FEW-Raw Features	-3.428	-5.652	-1.205	0.00119
2-d polynomial-Raw Features	-1.121	-3.345	1.102	0.45463
FEW-2-d polynomial	-2.308	-4.531	-0.084	0.04011

significant differences between FEW and raw features ($p = 0.0019$), FEW and 2-d polynomial ($p = 0.0401$), and no significant difference between 2-d polynomial and raw features ($p = 0.4546$).

6 Discussion

The results suggest that FEW can provide improved model predictions by learning a data representation for several problems and ML methods. Research questions remain, for example: how can FEW be improved when paired with specific learners? and: how can the number of features be more precisely controlled? Concerning the first question, it is highly likely that FEW could be more tailored to each learner with which it is paired, in the following ways: (i), by changes to the fitness function for each engineered feature; (ii), by adjusting the GP parameters of FEW for each ML method; (iii), by biasing the GP operators used to construct features such that they produce individuals more likely to benefit a given learner. For example to address (i), it may be advantageous to use an entropy criterion for fitness when FEW is paired with DT in a classification problem. Doing so is not straightforward in combination with ϵ-lexicase survival, due to the de-aggregated fitness values that lexicase methods expect when selecting individuals. One solution is to sample the entropy values over subsets of the training cases.

Concerning the second question, it would also be advantageous to de-couple the number of engineered features from the population size without having to introduce more parameters as was necessary in [1]. The motivation for de-coupling the two is that they form a trade-off: fewer features create a more readable model, yet a smaller population size restricts the sampling of the search space. With regularization available in many ML methods, it is tempting to use a large population size, although this results in longer run-times. All this must be considered in the context of the goals of the problem at hand. Ideally the population size would adapt automatically to fit the needs of the problem. There is some precedent for dynamic population size management in GP that may be applicable in this regard [24,29].

As a final comment, FEW deserves a treatment on problems other than regression. For classification problems, a fitness function for continuous-valued features must be analyzed, since their outputs do not explicitly map to categorical labels. In addition, the options for engineered features may include boolean transformations for certain problems. FEW should be tested with popular ML methods in that domain, notably logistic regression with ℓ_1 regularization.

7 Conclusions

In this paper, we introduced FEW, a GP-based feature learning method that interfaces with Scikit-learn suite of standard ML methods to improve data representation, a pressing issue in ML today. FEW represents engineered features as individuals in a population which are used in conjuction by the chosen ML algorithm to create a model. In order to evolve complimentary features, a new survival technique, ϵ-lexicase survival, was presented that maintains a population of un-correlated features by pressuring them to perform well on unique subsets of training cases. In comparison to polynomial feature expansion, FEW is better able to improve model predictions for the problems and ML methods tested.

Acknowledgments. This work was supported by the Warren Center for Network and Data Science at the University of Pennsylvania, as well as NIH grants P30-ES013508, AI116794 and LM009012.

References

1. Arnaldo, I., O'Reilly, U.M., Veeramachaneni, K.: Building predictive models via feature synthesis, pp. 983–990. ACM Press (2015)
2. Bengio, Y., Courville, A., Vincent, P.: Representation learning: a review and new perspectives. IEEE Trans. Pattern Anal. Mach. Intell. **35**(8), 1798–1828 (2013)
3. De Melo, V.V.: Kaizen programming, pp. 895–902. ACM Press (2014)
4. Foster, D., Karloff, H., Thaler, J.: Variable selection is hard. In: Proceedings of The 28th Conference on Learning Theory, pp. 696–709 (2015)
5. Friedman, J., Hastie, T., Tibshirani, R.: The elements of statistical learning. Springer series in statistics, vol. 1. Springer, Berlin (2001)
6. Guyon, I., Elisseeff, A.: An introduction to variable and feature selection. J. Mach. Learn. Res. **3**, 1157–1182 (2003)
7. Harrison, D., Rubinfeld, D.L.: Hedonic housing prices and the demand for clean air. J. Environ. Econ. Manage. **5**(1), 81–102 (1978)
8. Helmuth, T., Spector, L., Matheson, J.: Solving uncompromising problems with lexicase selection. IEEE Trans. Evol. Comput. **PP**(99), 1–1 (2014)
9. Iba, H., Sato, T.: Genetic Programming with Local Hill-Climbing. Tech. Rep. ETL-TR-94-4, Electrotechnical Laboratory, 1-1-4 Umezono, Tsukuba-city, Ibaraki, 305, Japan (1994). http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/Iba_1994_GPlHC.pdf
10. Icke, I., Bongard, J.C.: Improving genetic programming based symbolic regression using deterministic machine learning. In: IEEE Congress on Evolutionary Computation (CEC), 2013, pp. 1763–1770. IEEE (2013)
11. Kamath, U., Lin, J., De Jong, K.: SAX-EFG: an evolutionary feature generation framework for time series classification, pp. 533–540. ACM Press (2014)
12. Kommenda, M., Kronberger, G., Winkler, S., Affenzeller, M., Wagner, S.: Effects of constant optimization by nonlinear least squares minimization in symbolic regression. In: Blum, C., Alba, E., Bartz-Beielstein, T., Loiacono, D., Luna, F., Mehnen, J., Ochoa, G., Preuss, M., Tantar, E., Vanneschi, L. (eds.) GECCO 2013 Companion, pp. 1121–1128. ACM, Amsterdam (2013)

13. La Cava, W., Danai, K., Spector, L., Fleming, P., Wright, A., Lackner, M.: Automatic identification of wind turbine models using evolutionary multiobjective optimization. Renew. Energy Part 2 **87**, 892–902 (2016)
14. La Cava, W., Spector, L., Danai, K.: Epsilon-Lexicase Selection for Regression, pp. 741–748. ACM Press (2016)
15. Liskowski, P., Krawiec, K., Helmuth, T., Spector, L.: Comparison of semantic-aware selection methods in genetic programming. In: Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation, GECCO Companion 2015, pp. 1301–1307. ACM, New York (2015)
16. McConaghy, T.: FFX: fast, scalable, deterministic symbolic regression technology. In: Riolo, R., Vladislavleva, E., Moore, J.H. (eds.) Genetic Programming Theory and Practice IX. Genetic and Evolutionary Computation, pp. 235–260. Springer, New York (2011)
17. Muharram, M., Smith, G.D.: Evolutionary constructive induction. IEEE Trans. Knowl. Data Eng. **17**(11), 1518–1528 (2005)
18. Muharram, M.A., Smith, G.D.: The effect of evolved attributes on classification algorithms. In: Gedeon, T.T.D., Fung, L.C.C. (eds.) AI 2003. LNCS (LNAI), vol. 2903, pp. 933–941. Springer, Heidelberg (2003). doi:10.1007/978-3-540-24581-0_80
19. Muharram, M.A., Smith, G.D.: Evolutionary feature construction using information gain and gini index. In: Keijzer, M., O'Reilly, U.-M., Lucas, S., Costa, E., Soule, T. (eds.) EuroGP 2004. LNCS, vol. 3003, pp. 379–388. Springer, Heidelberg (2004). doi:10.1007/978-3-540-24650-3_36
20. Olson, R.S., Bartley, N., Urbanowicz, R.J., Moore, J.H.: Evaluation of a tree-based pipeline optimization tool for automating data science. arXiv preprint (2016). http://arxiv.org/abs/1603.06212
21. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
22. Redmond, M., Baveja, A.: A data-driven software tool for enabling cooperative information sharing among police departments. Eur. J. Oper. Res. **141**(3), 660–678 (2002)
23. Spector, L.: 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, pp. 401–408 (2012)
24. Tan, K.C., Lee, T.H., Khor, E.F.: Evolutionary algorithms with dynamic population size and local exploration for multiobjective optimization. IEEE Trans. Evol. Comput. **5**(6), 565–588 (2001)
25. Tibshirani, R.: Regression shrinkage and selection via the lasso. J. R. Stat. Soc. Ser. B (Methodological) **58**, 267–288 (1996)
26. Topchy, A., Punch, W.F.: Faster genetic programming based on local gradient search of numeric leaf values. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), pp. 155–162 (2001)
27. Torres-Sospedra, J., Montoliu, R., Martnez-Us, A., Avariento, J.P., Arnau, T.J., Benedito-Bordonau, M., Huerta, J.: UJIIndoorLoc: A new multi-building and multi-floor database for WLAN fingerprint-based indoor localization problems. In: 2014 International Conference on Indoor Positioning and Indoor Navigation (IPIN), pp. 261–270. IEEE (2014)
28. Tsanas, A., Xifara, A.: Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. Energy Build. **49**, 560–567 (2012)

29. Vanneschi, L., Cuccu, G.: A study of genetic programming variable population size for dynamic optimization problems. In: International Conference on Evolutionary Computation (ICEC 2009), pp. 119–126. Madeira, Portugal (2009)
30. Vladislavleva, E., Smits, G., Hertog, D.: Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. IEEE Trans. Evol. Comput. **13**(2), 333–349 (2009)
31. White, D.R., McDermott, J., Castelli, M., Manzoni, L., Goldman, B.W., Kronberger, G., Jakowski, W., O'Reilly, U.M., Luke, S.: Better GP benchmarks: community survey results and proposals. Genet. Program. Evolvable Mach. **14**(1), 3–29 (2012)