Building Predictive Models via Feature Synthesis

Ignacio Arnaldo
Massachusetts Institute of Technology, CSAIL
Cambridge, MA
iarnaldo@mit.edu

Una-May O'Reilly
Massachusetts Institute of Technology, CSAIL
Cambridge, MA
unamay@csail.mit.edu

Kalyan Veeramachaneni
Massachusetts Institute of Technology, CSAIL
Cambridge, MA
kalyan@csail.mit.edu

ABSTRACT

We introduce Evolutionary Feature Synthesis[1] (EFS), a regression method that generates readable, nonlinear models of small to medium size datasets in seconds. EFS is, to the best of our knowledge, the fastest regression tool based on evolutionary computation reported to date. The *feature search* involved in the proposed method is composed of two main steps: *feature composition* and *feature subset selection*. EFS adopts a bottom-up feature composition strategy that eliminates the need for a symbolic representation of the features and exploits the variable selection process involved in pathwise regularized linear regression to perform the feature subset selection step. The result is a regression method that is competitive against neural networks, and outperforms both linear methods and Multiple Regression Genetic Programming, up to now the best regression tool based on evolutionary computation.

Categories and Subject Descriptors

I.2.2 [**Artificial intelligence**]: Automatic Programming

Keywords

Regression; Feature Synthesis; Feature Subset Selection

1. INTRODUCTION

Recent methods such as Multiple Regression Genetic Programming (MRGP) [2], Behavioral Genetic Programming (BGP) [13], and Kaizen Programming (KP) [4] explore the combination of Genetic Programming based symbolic regression with deterministic machine learning methods. Tree-based GP is used to form complex expressions and machine learning tools fine-tune the models (MRGP and KP), guide selection (KP and BGP), or determine appropriate crossover points (BGP). While these methods outperform state-of-the-art GP approaches, their running time remains a major drawback that arguably limits their adoption by the wider machine learning community.

Significant training time reduction can be achieved by adopting a search paradigm where the unit of selection is the *feature*, not the model itself, and where the goal is to evolve a small population of features that collectively allows a good model to be obtained. This method reduces bloat, population size, and number of evaluations, and introduces major conceptual changes in the evolutionary process [4]. For instance, with this approach, the *importance* of a given feature is relative to the other features present in the population. This represents a notable difference with typical evolutionary search processes, where the goal is to find individuals (features in this case) that perform well independently. In this paper, we introduce an efficient and robust feature subset selection method, i.e. a strategy for selecting a subset of *important features* from the population for use in model construction, that is critical for the success of the *feature synthesis* paradigm.

Although feature subset selection has been widely investigated (see [9] and therein), it remains a challenging problem. On the one hand, the quality of the subsets depends on the chosen model and learning algorithm. On the other, it can be computationally expensive to explore many subsets. To alleviate the computational burden, there are *stepwise* alternatives that start with an empty feature set and greedily add the feature that most improves performance. Another alternative for feature selection is the use of regularization methods. Regularization methods incorporate a penalty for complexity that results in sparse models, which in turn can be exploited for feature selection.

Evolutionary Feature Synthesis (EFS) is composed of two main steps: *feature composition* with variation operators and *feature subset selection*[2] via a pathwise regularization method [5]. Special focus is dedicated to 1) the development of a robust feature subset selection strategy, and 2) to motivate the design decisions that lead to an efficient feature search. The output of EFS is a readable model, expressed as a linear combination of the *evolved population of features*. Our approach provides fast learning times and yields competitive accuracy in the set of studied benchmarks when compared to neural networks, and outperforms both MRGP and linear methods. To the best of our knowledge, EFS is the first evolutionary method that learns competitive, yet readable models of small to medium sized datasets in seconds.

[1]Project website: http://flexgp.github.io/efs

GECCO '15, July 11–15, 2015, Madrid, Spain

ISBN 978-1-4503-3472-3/15/07…$15.00

DOI: http://dx.doi.org/10.1145/2739480.2754693

[2]Not to be confused with the concept of training subset selection introduced by Gathercole and Ross in [8]

The paper is organized as follows. Section 2 introduces the machine learning methods at the core of EFS and Section 3 provides an overview of the method. In Section 4, we explain the feature composition step while in Section 5, we discuss feature subset selection techniques. Section 6 describes the experimental setup and we present the results in Section 7. Finally, Section 8 presents a review of related work and we conclude in Section 9.

2. BACKGROUND

We introduce the machine learning techniques at the core of our method, namely multiple linear regression, basis expansions, and pathwise LASSO regression (see [10]).

2.1 Multiple Linear Regression

Let us consider a dataset $(X^1, Y^1), \ldots, (X^n, Y^n)$ with univariate response variable Y and a $p-$dimensional vector of variables X. Multiple Linear Regression consists in finding the vector of coefficients $\hat{\beta}$ such that:

$$\hat{\beta} = \underset{\beta \in \mathbb{R}^p}{\operatorname{argmin}} ||Y - \sum_{i=1}^{p} \beta_i X_i||_2^2 \quad (1)$$

2.2 Basis Expansions

Linear Regression models can be extended with basis expansions to model nonlinear relations between the response variable Y and the variables X. We model a linear basis expansion in X as follows:

$$f(X) = \sum_{m=1}^{M} \beta_m h_m(X) \quad (2)$$

where $h_m(X) : \mathbb{R}^p \to \mathbb{R}$ is the mth transformation of X, $m = 1, \ldots, M$. In this context, the space determined by the basis functions $h_m, m = 1, \ldots, M$ is referred to as *feature space*, while the variables $X_j, j = 1, \ldots, p$ are referred to as *original variables*. The resulting regression problem is formulated as:

$$\hat{\beta} = \underset{\beta \in \mathbb{R}^m}{\operatorname{argmin}} ||Y - \sum_{m=1}^{M} \beta_m h_m(X)||_2^2 \quad (3)$$

Examples of simple transformations are $h(X) = cos(X_j)$, $h(X) = sin(X_j)$, $h(X) = log(X_j)$ [10]. One can also use functions involving several inputs, such as $h(X) = X_i \times X_j$. It is worth noting that the use of basis functions can make the dimensionality of the problem increase significantly. It is for this reason that the use of basis expansions is usually coupled with regularization methods.

2.3 Pathwise LASSO Regression

Least Absolute Shrinkage and Selection (LASSO) regression [18] is an extension of multiple linear regression where the regularization term $\|\beta\|_1$, the L1-norm of the weight vector, is added to the formulation of the problem, resulting:

$$\hat{\beta}^{\lambda} = \underset{\beta \in \mathbb{R}^m}{\operatorname{argmin}} ||Y - \sum_{m=1}^{M} \beta_m h_m(X)||_2^2 + \lambda \sum_{m=1}^{M} |\beta_m| \quad (4)$$

where $\lambda \in \mathbb{R}^+$ is the regularization coefficient. The LASSO penalty results in models with many coefficients close to zero, and a small subset larger and nonzero. Regularization is therefore helpful in problems where p is large, and is widely used for variable selection.

We resort to an efficient implementation of LASSO [6] based on the pathwise coordinate descent method introduced in [5]. This method computes the weights $\hat{\beta}$ for a decreasing sequence of regularization coefficients $\lambda \in \Gamma$. Note that the weights β^{λ} will vary for different values of λ.

3. METHOD OVERVIEW

EFS combines concepts of evolutionary computation with a state-of-the-art implementation of regularized linear regression. Our method relies on the assumption that the functions $h_m(X)$ introduced in Eq.(2) can be optimized by evolutionary computation. We design a population-based search where the unit of selection is *the feature*, as opposed to most related works where *models* are searched. In the following, we describe the steps involved in EFS (also depicted in Figure 1):

Initial Population The *population of features* is seeded with the functions $h_m(X) = X_m$, $m = 1, \ldots, p$, i.e. with the original variables of the problem.

Then, EFS executes an optimization loop composed of three steps: model generation, feature composition, and feature subset selection.

Model generation A first LASSO run obtains a linear combination of the features of the current population. The generated model is archived if it reduces the error of the best model found in previous iterations.

Feature composition In this step detailed in Section 4, variation operators augment the population with new features.

Feature subset selection We perform a second LASSO run on the augmented population of features and exploit the information generated during the regression process to estimate *feature importance*. The estimated *importance* is used to select the subset of the extended population that will be passed to the next generation. More details are provided in Section 5.

Stop criteria The process stops if the best model is not replaced in a preset number of iterations, or if a timeout is reached.

It is worth mentioning that, because the population of features is seeded with the original variables, our method guarantees an accuracy with respect to training data at least as good as that of the LASSO fit.

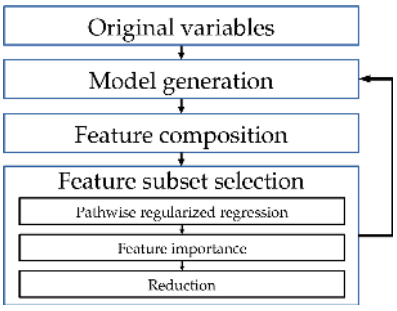


Figure 1: Steps involved in evolutionary feature synthesis.

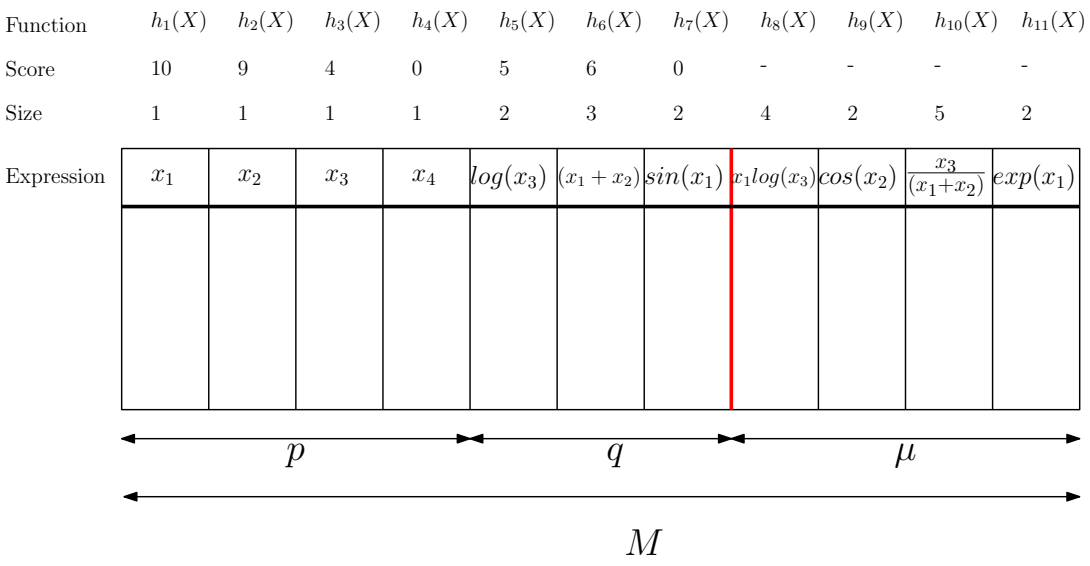


Figure 2: Feature Composition. Every generation, μ new features are composed by applying unary or binary operators to the current population of features. The size of the current population is given by $p+q$, where p is the dimensionality of the data and q is the number of composed features surviving from the previous generation.

4. FEATURE COMPOSITION

This step is inspired by the crossover and mutation steps performed in tree-based Genetic Programming and can be seen as generating new columns of the covariates matrix.

4.1 Bottom-up generation of new features

A set of μ features are composed by applying unary or binary mathematical operators to the existing features. The system admits the following set of operators:

$$O = \{+, -, *, /, exp, log, sqrt, square, cube, sin, cos\} \quad (5)$$

A tournament selection based on a precomputed *feature score* (see Section 5) is used to choose between $h_1, \ldots, h_{p+q}$, the features that will be combined to generate new features. We comment the example depicted in Figure 2. The values of the parameters are as follows:

- $p = 4$: the dimensionality of the problem,
- $q = 3$: the number of features added to the model,
- $\mu = 4$: the features composed in a given iteration,
- $M = p + q + \mu$: the population size.

We enumerate below the operations performed to the features $h_1, \ldots, h_7$ to generate the features $h_8, \ldots, h_{11}$:

- h_8: is a composition of h_1 and h_5 with the op. $*$,
- h_9: is a composition of h_2 with the op. *cos*,
- h_{10}: is a composition of h_3 and h_6 with the op. $/$,
- h_{11}: is a composition of h_1 with the op. *exp*.

4.2 Control for feature complexity

We set a maximum size for the generated features. We define the size of a feature as the count of *original variables* and *operators* appearing in the feature. For instance, the size of the features x_1, $log(x_1)$, and $\frac{x_3}{(x_1+x_2)}$ is respectively 1, 2, and 5. Newly generated features are discarded if their size is greater than the preset maximum. Let h_{p_1}, h_{p_2} be the features combined to generate a given feature h_i. Then, the size of h_i is given by:

$$size(h_i) = \begin{cases} size(h_{p_1}) + size(h_{p_2}) + 1, & \text{if binary } op \\ size(h_{p_1}) + 1, & \text{if unary } op \end{cases} \quad (6)$$

In this work, the maximum size is set to 5 in order to obtain readable features that can be interpreted by domain experts. This technique is also used to control model complexity in tree-based Genetic Programming. In fact, the proposed complexity measure can be seen as the size of the expression tree representing a given feature.

4.3 Differences with Genetic Programming

It is important to stress the following differences with respect to the crossover and mutation steps employed in tree-based Genetic Programming:

Vectorial operations vs. postponed evaluation: Given two existing features and an operator, the generation of a new feature is a vectorial (column-wise) operation. Vectorial operations are efficient and suitable for parallel execution. On the other hand, in tree-based Genetic Programming, crossover and mutation only modify the tree representation of the models. Later, in the evaluation step, each of the modified models needs to be traversed n times, where n is the number of exemplars in the dataset.

No symbolic representation: The values of a given feature are computed on the fly when the feature is first generated. Therefore, no symbolic representation is needed. Note that, however, a string representation of the generated feature is stored for logging purposes.

5. METHODS FOR FEATURE SELECTION

We introduce two strategies that run pathwise LASSO regression on the features of the population, and that exploit the generated information to guide feature selection.

5.1 Strategy 1: Flexible Model Size

Given a population of features P with size $|P| = M$, the goal is to select a subset $P' \subset P$ such that $|P'| \leq M$. As a first approach, we simply analyze the weights of the model obtained with LASSO on the augmented population and discard the features with zero weights. The steps involved in this first approach are described below:

Step 1: We run LASSO via pathwise coordinate descent (see [5]) on the extended population. As a result, we obtain the weight vectors β^{λ}, $\forall\lambda \in \Gamma$, i.e. for a set of decreasing regularization coefficients. Note that the coefficient of multiple correlation R^2 of each weight vector can be retrieved at no additional cost.

Step 2: We select the weights that maximize the coefficient of multiple correlation R^2. That is, we select the weights β^m such that $R^2_m \geq R^2_\lambda, \forall\lambda \in \Gamma$.

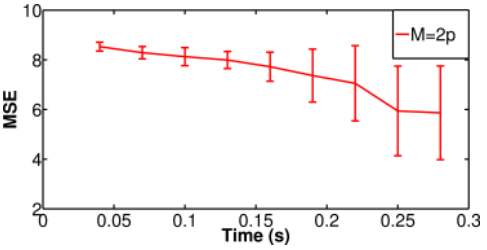
Step 3: We keep the original variables and the composed features with non-zero weights, and discard the rest. Note that the number of zero weights is not fixed beforehand, and thus the number of selected features will vary from an iteration to the other. The surviving features will be included in a new model and will be used to compose new features in the next generation.

This method presents one major drawback since, in the case where all the features have non-zero weights, no reduction occurs and the process stops prematurely.

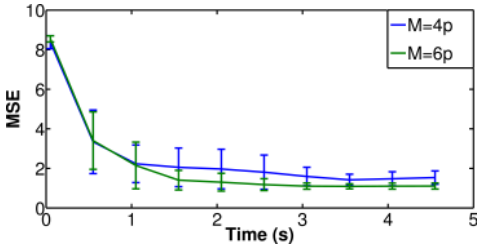
As a preliminary test, we benchmark this approach with the heating energy efficiency (ENH) regression problem introduced in [19]. The characteristics of this dataset are summarized in Table 3. We explore three different population sizes $M = 2p = 16$, $M = 4p = 32$, and $M = 6p = 48$, where p is the number of original variables of the problem (in this case $p = 8$) and perform 20 runs for each of the three configurations.

Figure 3 shows the mean squared error (MSE) over time of the models obtained with the three population sizes. In the case where $M = 2p$ (Figure 3a), it can be seen that, although an average error of 5.866 is achieved, the method presents a great variance over the 20 runs. The variance is the consequence of premature stops of the search, observed when all the features of the population are assigned non-zero weights. In fact, all the 20 runs stopped in less than 0.44s and executed an average of only 13 generations. On the other hand, when $M = 4p$ or $M = 6p$ (Figure 3b), the average MSE is reduced over time, and achieves respectively 1.536 and 1.107 in less that 5 seconds. The variance seems to decrease for increasing population sizes since the case where $M = 6p$ presents the lowest variability between runs.

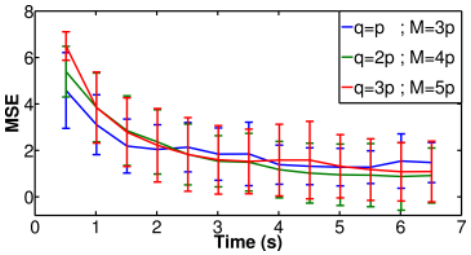
These results show that our method is capable of finding features leading to a better accuracy on the training set. However, this error reduction is achieved at the expense of an increased complexity of the final model. Next, we explore an extension of this approach that achieves simpler models.



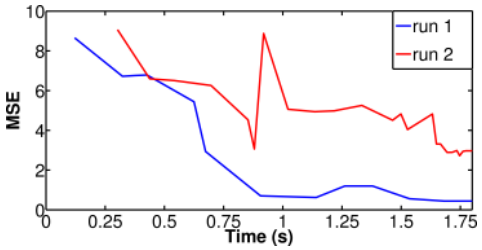
(a) Flexible model size: M=$2p$



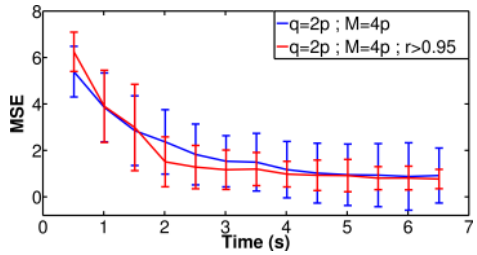
(b) Flexible model size: M=$4p$ vs M=$6p$



(c) Fixed model Size: (q=p;M=$3p$) vs (q=$2p$;M=$4p$) vs (q=$3p$;M=$5p$)



(d) Fixed model size: 2 example runs with (q = p; M = $3p$)



(e) Fixed model size: (q=$2p$;M=$4p$) vs correlation filtering with (q=$2p$;M=$4p$;r>0.95)

Figure 3: Mean Squared Error (MSE) on training data vs time (seconds) with the different studied subset selection strategies. The plots are obtained by averaging 20 runs on the heating energy efficiency dataset. The error bars show the standard deviation.

5.2 Strategy 2: Fixed Model Size

We set a maximum model size, that is, a maximum number of features included in the model. Note that, as in the previous strategy, we force the survival of the original variables of the problem.

Let P be a population of features with size $|P|=M$, then the goal is to select a subset of $P' \subset P$ such that $|P'|<M$ and $|P'|=p+q$, where p is the number of original variables of the problem and q is fixed and represents the number of composed features added to the model. This strategy results in a model including at most $p+q$ features and $\mu=M-(p+q)$ discarded features.

In this case, looking at non-zero weights is not enough, since the number of composed features with non-zero weights can be greater than the preset q. Instead, we perform a complete ranking of the M features of the population according to an estimation of their importance. The proposed *importance measure* captures the number of times a given feature appears in the model for different values of the regularization coefficient λ, as well as the accuracy of the models in which it appears:

Step 1: As in Strategy 1, we run LASSO via pathwise coordinate descent on the extended population.

Step 2: We estimate the *importance* of each feature h_j as:

$$importance(h_j) = \sum_{\lambda_i \in \Gamma} score(j, \beta^{\lambda_i}) \quad (7)$$

$$score(j, \beta^{\lambda_i}) = \begin{cases} R^2_{\lambda_i} \text{ if } \beta_j^{\lambda_i} \neq 0 \\ 0 \text{ otherwise} \end{cases} \quad (8)$$

where $R^2_{\lambda_i}$ and $\beta_j^{\lambda_i}$ are respectively the coefficient of multiple correlation and the weight of h_j obtained for λ_i.

Step 3: We rank the features according to the estimated *importance*. The p original variables and the top q composed features survive and will be used in the next generation to compose new features. The rest of the features are discarded.

We perform a preliminary test with the heating energy efficiency dataset also used to benchmark the previous selection strategy. Since the goal is to obtain simpler models, we explore the following configurations:

- $q=p; M=3p$: the model includes at most $2p$ features
- $q=2p; M=4p$: the model includes at most $3p$ features
- $q=3p; M=5p$: the model includes at most $4p$ features

Table 1 shows a model obtained with $q=2p$ and $M=4p$. Note that we verified that none of the 19 features of the presented model are constants, and that removing any of the features has an impact on the model's error.

Figure 3c shows the MSE over time of the models obtained with the three configurations. In all cases, the error is reduced over time. The first setup ($q=p; M=3p$) depicted in blue yields the highest MSE (1.278). The second configuration (green), minimizes the error of the resulting model (0.875) and also includes less features than the third (red) configuration (MSE=1.081). Therefore, we fix the parameters $q=2p$ and $M=4p$ going forward.

The performed analysis reveals a great variance of the error over the different runs. This variance is due to the fact

TRAIN SET MSE: 0.251

```
-    40.763
+     0.328    * X6
-    20.690    * X7
-     0.838    * X8
-   150.026    * (log(cos(cube(sin X4))))
+     2.528    * (sin(sin(sin(sin X2))))
-     0.434    * (sin(cube(* X5 X3)))
+     1.313    * (sin(cos(* X5 X3)))
+ 1.280E-14    * (square(square(* X2 X5)))
-     0.066    * (sin(cos(+ X8 X5)))
+    12.014    * (sqrt(divide X5 (cos X7)))
+     8.156    * (exp(sqrt(* X1 X7)))
+     0.361    * (- (square(sin X8)) X6)
+    55.787    * (sqrt(divide(sqrt X8) X4))
-    12.661    * (cos(sin(divide X4 X1)))
-     0.487    * (sin X8)
-     3.417    * (sin(sin(divide X4 X1)))
+     8.834    * (sqrt(sqrt(square(cos X2))))
+    19.890    * (cos(square(square(cos X2))))
+     0.094    * (*(* X1 X7)X3)
```

Table 1: Model reached in 8.648 seconds for the ENH dataset with $q=2p; M=4p$. The maximum feature size is set to 5.

FEATURE	RANK	IMPORT.
(-(sin(cos X3))X5)	1st	87.3
(div(cube(cos X4))X1)	2nd	86.1
(* X3(* X7 X5))	3rd	83.8
(sqrt(sqrt(sqrt(sqrt X8))))	4th	67.1
(cos(cos(cube(sin X3))))	5th	64.4
(square X6)	6th	55.2
(sin(square X6))	7th	49.7
(cube(exp(sin X2)))	8th	49.7
----------	----	----
(cos X2)	9th	43.7
(log(exp(sin X2)))	10th	42.3
(exp(cos X2))	11th	34.7
(div X1(square X6))	12th	30.9
(exp(sin X2))	13th	30.0
(cos X1)	14th	27.6
(exp X8)	15th	24.1

Table 2: Features present in the extended population and their rank and score. The two features in red (9th and 11th) are discarded because they are highly correlated ($r=0.975$).

that model error is not systematically reduced after each selection step. Figure 3d shows two runs illustrating this issue. The first (run 1) is an example of appropriate behavior of the search where the MSE of the generated model decreases over time. On the other hand, in the second run there is a spike at 0.92s, corresponding to the 7^{th} generation, where the MSE on the training set jumps from 3.064 to 8.878.

To understand the causes of such undesirable behavior, we report in Table 2 the ranking of the extended population of features at the end of the 7^{th} generation. Note that, in this particular case, 8 features need to be selected. We highlight two features, $cosX2$ (9^{th}) and $exp(cosX2)$ (11^{th}), that are discarded although their estimated importance shows that they are relevant. These two features are highly correlated, with a Pearson correlation coefficient of 0.975. This shows that, because these two features are correlated, their importance is split between them and thus neither gets picked, while the desired behavior would be to pick one and discard the other. Next, we further improve our method to alleviate the issues caused by correlated variables.

5.3 Strategy 3: Correlation Filtering

A complete correlation analysis of the M features of the population is expensive since M^2 correlations need to be

calculated, and each of the computations requires a pass over the data. To avoid the extra computational cost, we only check the Pearson correlation coefficient r of a given feature with its parent(s) (see Section 4): the feature in question is discarded if the correlation with any of its parents is higher than a given threshold t. We implement this filter during the feature composition step, updating the partial sums necessary for the computation of r as the values of the new feature are computed. This way, no extra passes over the data are required.

We set the correlation threshold to $t = 0.95$ and the parameters $q = 2p$ and $M = 4p$ and benchmark our method with the same heating energy dataset used previously. Figure 3e shows the MSE over time of the models obtained without correlation filtering and with the filter $r > 0.95$. The average MSE obtained with both approaches is almost identical. The variance over the 20 runs, however, is reduced when the filtering strategy is adopted. Therefore, the correlation check, done at a negligible cost, seems to improve the robustness of our method against correlated features.

With this final design, we turn to analyze whether EFS is competitive against state-of-the-art regression methods in terms of accuracy and running time, while generating readable symbolic models.

6. EXPERIMENTAL SETUP

In this section, we introduce the datasets and regression approaches used to benchmark EFS.

6.1 Datasets

We consider datasets where the variables correspond to physical measurements. Such benchmarks are suitable to analyze whether EFS is capable of finding higher-level features that result in a more accurate model. Table 3 summarizes the regression problems used in this study. The datasets ENH and ENC (energy efficiency of heating and cooling loads) come from simulations [19] while the NOx emissions dataset is a collection of real power plant data [20]. In both the red and white wine quality datasets, the original variables are physical measurements, but the dependent variable is a subjective grade, expressed on a scale from very bad (0) to excellent (10). Finally, the Million Song Dataset (MSD) year prediction challenge, introduced in [3], is a regression problem in which the goal is to predict the release year of a large set of songs. The size and dimensionality of the dataset are challenging, since it is composed of 515K songs, each described with 90 features and a year label.

6.2 Compared Approaches

We introduce the compared regression methods:

Multiple Linear Regression (MR): We obtain a linear model of the data using the least squares approach.

Dataset	p	Exemplars		
		Training	Test	Total
ENC	8	512	256	768
ENH	8	512	256	768
NOX	18	4017	1210	5227
WIR	11	1066	533	1559
WIW	11	3265	1633	4898
MSD	90	413124	102440	515564

Table 3: Benchmark datasets.

LASSO: The same implementation of LASSO embedded in EFS is used to perform L1-regularized linear regression.
Vowpal Wabbit (VW): Vowpal Wabbit [14] is a fast out-of-core online learning algorithm based on sparse gradient descent. VW generates linear models of large datasets (that might not fit in RAM) in minimal time.
Feed Forward Neural Networks (FFNN): We consider a configuration with one hidden layer of 10 neurons. We employ Matlab's Neural Network Toolbox [16] to train the FFNNs. The networks are trained via backpropagation with the Levenberg-Marquardt optimization.
Multiple Regression Genetic Programming (MRGP): MRGP is a hybrid method that combines tree-based Genetic Programming with LASSO [1, 2]. MRGP outperforms both multiple linear regression and traditional GP-based symbolic regression methods. We exploit MRGP's parallelism by executing it in a 4-threaded fashion.
EFS: We run EFS with the following parameters: $q = 2p$ and $M = 4p$ (see Section 5.2). Additionally, we adopt the correlation filtering strategy ($r > 0.95$) presented in Section 5.3.

7. RESULTS

We compare the mean squared error (MSE) and running time of the compared approaches. Note that we perform 20 runs of the algorithms that present a stochastic nature, that is, FFNN, MRGP, and EFS.

7.1 Comparison in terms of error

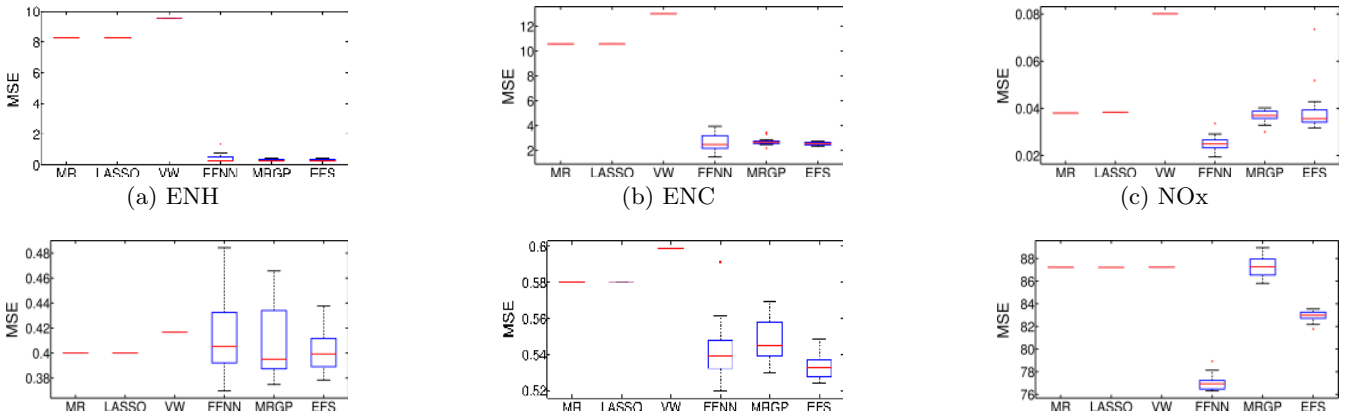
Table 4 reports the average test MSE achieved by the compared approaches for the different datasets and Figure 4 shows the boxplots generated with the 20 runs.

ENH: Nonlinear methods, i.e. FFNN, MRGP, and EFS clearly outperform the linear ones (MR, LASSO, and VW). In particular MRGP and EFS perform particularly well reaching an MSE of 0.314 and 0.317 respectively.
ENC: As in the previous case, FFNN, MRGP, and EFS significantly reduce the error of the linear methods. In this case, EFS presents the lowest MSE (2.548).
NOx: MRGP and EFS remain close to the linear methods in terms of accuracy. FFNN performs particularly well on this dataset, reaching an MSE of only 0.025.
WIR: This is the only dataset where the linear methods MR and LASSO, with an MSE of 0.400, outperform the rest of the methods. EFS remains close to the linear models with an MSE of 0.402.
WIW: Nonlinear methods improve the accuracy of MR or LASSO only marginally, reducing the MSE from 0.580 to 0.533 in the case of EFS.
MSD: MRGP obtains an MSE of 87.233, similar to that of the linear models. On the other hand, EFS obtains an error of 82.903 and FFNN further reduces it to 77.019.

Overall, FFNN and EFS present the lowest error, with an average rank of 2.333. They are followed by MRGP, LASSO, and MR with a rank of 3.167, 3.583, and 3.750 respectively. Finally, VW ranks 5.833 on average. It is worth mentioning that VW is sensitive to the parameters governing the learning process, and, according to our experience, it is possible to find a set of parameters that will achieve an accuracy close to that of MR.

	ENH		ENC		NOx		WIR		WIW		MSD		Rank	
	MSE	Time	MSE	Time	MSE	Time	MSE	Time	MSE	Time	MSE	Time	MSE	Time
MR	8.303	0.040	10.576	0.030	0.038	0.120	0.400	0.030	0.580	0.050	87.225	41.100	3.750	2.167
LASSO	8.282	0.142	10.584	0.154	0.038	0.319	0.400	0.162	0.580	0.224	87.217	17.409	3.583	2.833
VW	9.577	0.007	13.012	0.014	0.080	0.039	0.417	0.011	0.599	0.039	87.233	12.473	5.833	1.000
FFNN	0.400	1.036	2.649	1.042	0.025	3.186	0.414	0.622	0.542	1.415	77.019	2013.900	2.333	4.000
MRGP	0.314	60.000	2.703	60.000	0.037	60.000	0.407	60.000	0.548	60.000	87.313	3600.000	3.167	5.917
EFS	0.317	10.575	2.548	14.161	0.039	58.852	0.402	21.906	0.533	35.498	82.903	3600.000	2.333	5.083

Table 4: Testing set Mean Squared Error (MSE) and learning time in seconds of the compared regression techniques.



(a) ENH (b) ENC (c) NOx

(d) WIR (e) WIW (f) MSD

Figure 4: Testing set Mean Squared Error (MSE) of the compared regression techniques.

7.2 Comparison in terms of speed

As shown in Table 4, the studied methods are ranked by running time as follows.

VW: VW is the fastest method. It is worth mentioning that, in the case of the MSD dataset, it achieves an error close to that of MR in only 12.473 seconds.

MR and LASSO: The linear methods MR and LASSO rank second and third. The employed pathwise coordinate descent implementation is better suited for larger datasets. In fact, in the case of the largest dataset (MSD), MR runs for 41.100 seconds while LASSO needs 17.409 seconds to execute.

FFNN: FFNN are remarkably fast in the case of the ENH, ENC, NOx, WIR, and WIW datasets, taking between 0.622 and 3.186 seconds. The training time of FFNNs depends on the number of parameters that need to be learned. In the case of MSD, 910 parameters are learned, consuming an average of 2013.900 seconds.

EFS: Since we are interested in fast learning, we allowed EFS to run for one minute for the first five datasets, while the maximum running time for MSD was set to 60 minutes. EFS converged in 10.575 and 14.161 seconds for the ENH and ENC datasets respectively. In the case of the WIR (21.906s) and WIW (35.498s) datasets, it also finished before the established budget. On the other hand, the method did not converge in the established time for the NOx and MSD datasets.

MRGP: As for EFS, we allowed MRGP to run for 60 seconds for the ENH, ENC, NOx, WIR, and WIW datasets. For the larger MSD dataset, MRGP ran for one hour.

It is important to stress a key difference between MRGP and EFS and the rest of the approaches. The linear methods and FFNN focus the learning process on tuning the parameters of a model with a predefined structure. On the other hand, MRGP and EFS search for both the structure of the model and its parameters, resulting in longer training times which sometimes pay off with better accuracy while remaining acceptable in time cost.

7.3 A note on the complexity of the models

We comment the complexity of the retrieved models:

MR, LASSO, and VW: These linear methods obtain the simplest models. Given a dataset composed of p original variables, only $p+1$ weights need to be tuned during the learning process. In the case of LASSO and VW, the model is further simplified by means of a regularization penalty.

FFNN: The complexity of the FFNN is given by the structure of the network. In this case, although the designed network only counts one hidden layer of 10 neurons, the resulting model is hardly interpretable.

MRGP: MRGP models are expression trees coupled with a set of weights, each associated with one node of the tree. These models can be seen as highly complex basis expansion models (see Eq. (2)). Note that, in the case of a complete binary tree, the size of the resulting basis expansion model would be 2^h-1, where h denotes the height of the tree. The size of MRGP models can quickly explode for the standard maximum tree height $h=17$ used in tree-based GP.

EFS: EFS provides readable basis expansion models composed of the original variables and a set of composed features. We have determined that considering models of size p(original variables)$+2p$(additional features)$= 3p$ provides appropriate performance in practice. In this work, we limited the size of the composed features to be at most 5 (5 elements between variables and operators). We consider that this setting results in readable models that can be analyzed and interpreted by domain experts.

8. RELATED WORK

A large corpus of research uses GP to generate features; examples are [15], [7] and [12]. The combination of GP with LASSO is also exploited by the MRGP method [2].

EFS shares the *feature synthesis paradigm* with Kaizen Programming [4]. The main difference between the two methods is the feature subset selection strategy. EFS exploits pathwise regularized linear regression to avoid search stagnation, thus eliminating the need of restarting the run with a larger population size. Additionally, EFS does not require a symbolic representation of the features, yet it still provides flexibility to model nonlinear data.

Basis expansions allow domain experts to include knowledge of the problem to model nonlinear data. FFX [17] is a non-evolutionary technique that exploits this idea: it generates *many* basis functions and uses regularized linear regression to combine them. Icke *et al.* expand the FFX approach by running GP-based symbolic regression on the augmented space of features [11]. In these two works, the set of augmented features is fixed beforehand and is not explored during the learning process. It is the realization that the feature space can be efficiently searched with evolutionary computation that led us to develop the EFS method.

9. SUMMARY AND FUTURE WORK

We have introduced Evolutionary Feature Synthesis (EFS). EFS generates readable, nonlinear models of small to medium size datasets in seconds. In terms of accuracy, it is competitive against neural networks and outperforms both linear methods and MRGP, up to now the best regression tool based on evolutionary computation.

In future work, we will exploit the variable selection procedures involved in other machine learning algorithms such as decision tree induction algorithms. Also, the exploration of the feature synthesis paradigm in combination with the link functions used in generalized linear models, such as the logistic function, opens an wide area for exploration.

10. ACKNOWLEDGMENTS

This work was supported by the Li Ka Shing Foundation.

11. REFERENCES

[1] Multiple regression genetic programming in Java. http://flexgp.github.io/gp-learners/, 2014.

[2] I. Arnaldo, K. Krawiec, and U.-M. O'Reilly. Multiple regression genetic programming. In *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation*, GECCO '14, pages 879–886, New York, NY, USA, 2014. ACM.

[3] T. Bertin-Mahieux, D. Ellis, B. Whitman, and P. Lamere. The million song dataset. In *ISMIR 2011: Proceedings of the 12th International Society for Music Information Retrieval Conference, October 24-28, Miami, Florida*, pages 591–596, 2011.

[4] V. V. De Melo. Kaizen programming. In *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation*, GECCO '14, pages 895–902, New York, NY, USA, 2014. ACM.

[5] J. H. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2 2010.

[6] Y. Ganjisaffar. Lasso4j. https://code.google.com/p/lasso4j/, 2014.

[7] M. Garcia-Limon, H. J. Escalante, E. Morales, and A. Morales-Reyes. Simultaneous generation of prototypes and features through genetic programming. In *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation*, GECCO '14, pages 517–524, New York, NY, USA, 2014. ACM.

[8] C. Gathercole and P. Ross. Dynamic training subset selection for supervised learning in genetic programming. In Y. Davidor, H.-P. Schwefel, and R. Manner, editors, *Parallel Problem Solving from Nature, PPSN III*, volume 866 of *Lecture Notes in Computer Science*, pages 312–321. Springer Berlin Heidelberg, 1994.

[9] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, Mar. 2003.

[10] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning: data mining, inference and prediction*. Springer, 2nd edition, 2009.

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

[12] U. Kamath, J. Lin, and K. De Jong. SAX-EFG: An evolutionary feature generation framework for time series classification. In *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation*, GECCO '14, pages 533–540, New York, NY, USA, 2014. ACM.

[13] K. Krawiec and U.-M. O'Reilly. Behavioral programming: A broader and more detailed take on semantic GP. In *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation*, GECCO '14, pages 935–942, New York, NY, USA, 2014. ACM.

[14] J. Langford, L. Li, and T. Zhang. Sparse online learning via truncated gradient. *Journal of Machine Learning Research*, 10:777–801, June 2009.

[15] Y. Lin and B. Bhanu. Evolutionary feature synthesis for object recognition. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 35(2):156–171, May 2005.

[16] MathWorks. Neural network toolbox, 2014.

[17] T. McConaghy. FFX: Fast, scalable, deterministic symbolic regression technology. In R. Riolo, E. Vladislavleva, and J. H. Moore, editors, *Genetic Programming Theory and Practice IX*, Genetic and Evolutionary Computation, pages 235–260. Springer New York, 2011.

[18] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1994.

[19] A. Tsanas and A. Xifara. Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. *Energy and Buildings*, 49(0):560–567, 2012.

[20] F. Xue, R. Subbu, and P. Bonissone. Locally weighted fusion of multiple predictive models. In *International Joint Conference on Neural Networks, 2006. IJCNN '06.*, pages 2137–2143, 2006.