A method for regularization of evolutionary polynomial regressions

Francisco Coelho*

Departamento de Informática, Universidade de Évora, Rua Romão Ramalho 58, 7000-671 Évora, Portugal

João Pedro Neto*

University of Lisboa, Faculty of Sciences, BioISI– Biosystems & Integrative Sciences Institute, Campo Grande, 1749-016 Lisboa, Portugal

Abstract

While many applications require models that have no acceptable linear approximation, the simpler nonlinear models are defined by polynomials. The use of genetic algorithms to find polynomial models from data is known as Evolutionary Polynomial Regression. This paper introduces Evolutionary Polynomial Regression with Regularization, an algorithm that extends the EPR method with a regularization term to control polynomial complexity. The article also describes a set of experiences on common datasets that compare both flavors of EPR and other methods including Linear Regression, Regression Trees and Support Vector Regression.

The empiric conclusion of those experiments is that EPR with regularization is able to achieve better fitting than other non-ensemble methods and it needs less computation time than plain EPR.

Keywords: evolutionary polynomial regression, regularization, feature extraction

— GAs are repeatedly mentioned as an approach to search for locally optimal

*Corresponding author: Tel.: +351-919-006-379
Email addresses: fc@di.uevora.pt (Francisco Coelho), jpn@di.fc.ul.pt (João Pedro Neto)

parameters, this is simply incorrect and is a very surprising statement. GAs are NOT a local optimizer in any shape or form. Our intention was merely to note that GAs give no guarantee of finding a global optimum. We rephrase the mentioned sentences.

— What is the reference for the GA encoding used? It is ours. We stated this contribution more explicitly.

— The penalty term introduces is the main contribution of this work, this should be clearly stated in the text. It is basically a complexity based penalty which is really not so novel, there are even several cases in literature of penalty based GA functions. Please dig up further in the state of the art to back up better your claim of novelty. We included references to previous regularization in GA and better outlined the role and originality of our contribution.

— In some box plots only ten simulations are performed, in others 25. These numbers are too low. The number of samples was increased.

1. Introduction

With notable exceptions (*e.g.* neural networks) machine learning regression techniques produce linear models. The linearity assumption has many advantages including reduced computational complexity and strong theoretical framework. However nonlinearity is unavoidable in many application scenarios, specially those with phase transitions or feedback loops, so common in engineering, ecology, cybernetics and other areas. The kernel trick in Support Vector Machines (SVM) ([**? ? ?**]) alleviates this problem by allowing special non-linear transformations of the feature-space. The condition such transformations must meet is known as the *kernel trick*, $k(x, x') = \langle \varphi(x), \varphi(x') \rangle$, where φ is the feature-space transformation and $\langle \cdot, \cdot \rangle$ denotes inner product. The "trick" consists on computing the kernel $k(x, x')$ while avoiding the computation of the inner product and the transformations $\varphi(x), \varphi(x')$. A special case of polynomial transformation, the *polynomial kernel*, $k(x, x') = \langle x, x' \rangle^d$ is commonly used in regression and classification tasks with SVMs. However the kernel trick doesn't apply to general polynomial transformations.

Polynomials, one of the most studied subjects in mathematics, generalize linear functions and define, perhaps, the simplest and most used nonlinear models. For example, Polynomial Neural Networks [?] generalize the linear component of neural networks with a polynomial function. Applications include colorimetric calibration [?], explicit formulæ for turbulent pipe flows [?], computational linguistics [?] and more recently analytical techniques for cultural heritage materials [?], liquid epoxy moulding process [?], B-spline surface reconstruction [?], product design [?] or forecasting cyanotoxins presence in water reservoirs [?]. These examples not only illustrate the wide spectrum of applications but, additionally, each one uses, at some point, Genetic algorithms (GA).

Evolutionary algorithms, including GA, were, arguably, one of the hottest topics of research in the recent decades and with good reason since they outline an optimization scheme easy to conceptualize and with very broad application. If a nonlinear (or otherwise) model requires parameterization, GAs provide a simple and often effective approach to search for global optimal parameters (although no guarantee of global optimality can be given). Related research abound and spans from the 1950s seminal work of Nils Aall Barricelli [?] in the Institute for Advanced Study of Princeton to today's principal area of study for thousands of researchers, covered in hundreds of conferences, workshops and other meetings. Perhaps the key impulse to GAs came from John Holland's work and his book "Adaptation in Natural and Artificial Systems" [?].

One interesting variation of genetic algorithms, named *genetic programming* by John Koza [?], proposes the use of GAs to search the syntactic structure of complex functions. Syntactic structure search is also keen to the central ideas of deep learning [? ?], a subarea of machine learning actually producing quite promising results (*e.g.* in [?]). It is also related to the work presented in this paper in the sense that, unlike linear models that have a simple structure, $y = \sum_i \beta_i x_i$, nonlinear (in particular polynomial) models pose an additional structure search problem.

The idea of using GAs to find a polynomial regression is not new [? ? ?] but still generates original research [? ?]. The modern formulation of the use of GA to find polynomial models is known as Evolutionary Polynomial Regression (EPR) and systematization can be traced back to the work of Davidson, Savic and Walters [?]. Further developments include multi-objective optimizations [?].

Use of regularization/penalty functions is common practice in machine learning in general and has some applications in GA[?]. This paper describes an extension of the general EPR method to find a regularized polynomial regression of a given dataset. Herein optimal regression results from a cost function that accounts for both the root-mean-square (error) together with a novel regularization factor that penalizes over-fitting by polynomial complexity.

The next section describes the method's details and is followed by a presentation of some performance results. The last section draws some conclusions and points future research tasks.

2. Genetic Algorithms for Polynomials

This section starts with a brief introduction and outline of the evolutionary polynomial regression algorithm, EPR, and proceeds into core details as the encoding used to represent individual polynomial instances in the GA populations and the regularization of the cost function.

A usual representation of polynomials is through expressions of the form

$$p\left(x_1, \ldots, x_m\right) = \sum_i \theta_i q_i$$

where each $q_i = \prod_j x_j^{\alpha_{ij}}$ is a monomial, the exponents $\alpha_{ij} \in \mathbb{N}_0$ are non-negative integers and the coefficients $\theta_i \in \mathbb{R}$ are real valued. For example $p\left(x_1, x_2, x_3\right) = 2x_1 + x_2x_3 + \frac{1}{2}x_1^2x_3$ has monomials $q_1 = x_1, q_2 = x_2x_3$ and $q_3 = x_1^2x_3$, exponents $\alpha_{1,1} = 1, \alpha_{2,2} = 1, \alpha_{2,3} = 1, \alpha_{3,1} = 2, \alpha_{3,3} = 1$ and all other $\alpha_{ij} = 0$ and coefficients $\theta_1 = 2, \theta_2 = 1$ and $\theta_3 = 1/2$.

The exponents alone can be organized into a matrix $[\alpha_{ij}]$ that defines the monomial structure of the polynomial. For the example above the matrix representation of the monomials is

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 2 & 0 & 1 \end{bmatrix} \sim \begin{bmatrix} x_1 \\ x_2 x_3 \\ x_1^2 x_3 \end{bmatrix}$$

where each row defines a monomial and each column represents a variable. Changing the order of the rows doesn't change the polynomial whereas changing the order of the columns corresponds to changing the respective variables.

This partial representation of polynomials makes the problem of structure search very clear: except for the trivial cases, the number of possible monomials given n variables and a maximum joint degree d grows exponentially with either n or d. But more importantly, by separating the set of monomials from the coefficients, the polynomial regression problem can be naturally split into two subproblems:

1. For a given set of monomials $\mathcal{Q} = \{q_1, \ldots, q_k\}$ find the regression coefficients $\Theta = \{\theta_1, \ldots, \theta_k\}$ that minimize the error on a given dataset;
2. Find the fittest set of monomials, *i.e.* the polynomial that minimizes the error on the same dataset;

More precisely, concerning the first problem, let $\mathcal{D}$ be a dataset with n observations of variables $Y, X_1, \ldots, X_m$ and $\mathcal{Q} = \{q_1, \ldots, q_k\}$ a set of k monomial expressions over $X_1, \ldots, X_m$. Define the hypothesis[1]

$$h_{\Theta,\mathcal{Q}}(x_1, \ldots, x_m) \quad = \quad \sum_{j=1}^{k} \theta_j q_j |_{X_i = x_i, \forall 1 \leq i \leq m} \tag{1}$$

and let the error (as "cost") be

$$J_{\text{fit}}(\Theta; \mathcal{Q}, \mathcal{D}) \quad = \quad \sqrt{\frac{1}{n} \sum_{i=1}^{n} \left(y^{(i)} - h_{\Theta,\mathcal{Q}}\left(x_1^{(i)}, \ldots, x_m^{(i)}\right)\right)^2} \tag{2}$$

[1]The expression "$q|_{X=x}$" reads "*q with all instances of X replaced by x.*"

the usual root-mean-square (error) function. Now the first problem can be stated as: *Given a dataset $\mathcal{D}$ and a set of monomials $\mathcal{Q}$ find parameters Θ that minimize the cost $J_{fit}(\Theta; \mathcal{Q}, \mathcal{D})$.*

This is a simple linear regression problem obtained by expanding $\mathcal{D}$ with columns that replicate the monomials in $\mathcal{Q}$. The resulting dataset, $\mathcal{D} \cup \mathcal{Q}(\mathcal{D})$, adds the monomial transformations in $\mathcal{Q}$ to the original dataset $\mathcal{D}$. An alternative formulation would just replace $\mathcal{D}$ by $\mathcal{Q}(\mathcal{D})$. It turns out that the first formulation is a special case of the second (by including the variables in the monomial set) and has the potential for better error performance because it uses more features. — How is this known??? Is there a reference or some experimental results backing this claim or is it simply intuition??? We reformulated the previous sentence.

The second problem is treated in the GA setting: Let $\mathcal{D}$ be a dataset as above and $\mathcal{P}$ a set of polynomials. For each polynomial $p \in \mathcal{P}$ let $\mathcal{Q}_p$ be the set of monomials in p (without the coefficients) and define the minimization based fitness — say what? Do you mean something like a minimization based fitness?? Anti fitness seems odd. Adopted the reviewer suggestion.

$$\phi_p = \min_{\Theta} J_{\text{fit}}(\Theta; \mathcal{Q}_p, \mathcal{D}) \tag{3}$$

by solving the first problem. With a fitness of every instance, the GA genetic operators (usually mutation and crossover) evolve the population $\mathcal{P}$ until a reasonable approximation of a minimum is found. The properties of GAs and linear regression entail that Algorithm 1 converges to a polynomial that is a minimum of the fitness function, encapsulated in the error function J_{fit}.

Subsection 2.1 describes the encoding of individual polynomial instances as chromosomes and other parameters used in the GA implementation. The regularization of the cost function is discussed in subsection 2.2.

2.1. Polynomial Encoding

The specific encoding (representation) of a set of monomials is an important aspect in the implementation of EPR. The choice described below, developed

Algorithm 1 This EPR algorithm uses linear regression for the calculation of the error J and the space of polynomials is searched in the GAs iteration step. At exit the error of the fittest instance is bounded by ϵ or the maximum number of allowed iterations.

```
function EPR($D, pop_0, \epsilon, maxiter$)
    $pop \leftarrow pop_0$; $err \leftarrow 1.0 + \epsilon$
    while $err > \epsilon \wedge iterations < maxiter$ do
        $pop \leftarrow$ IterateGA($pop$)
        $pop \leftarrow$ Sort($pop, key = J$)        ▷ Sort population by regression error
        $err \leftarrow J$ (First($pop$))
    end while
    return First($pop$)
end function
```

by the authors for this algorithm, permits active and inactive monomials for regression purposes — is this an original contribution???? Otherwise it needs a reference. it is ours. The active (or inactive) state of a monomial might change through mutation or crossover. This simple mechanism enhances variation in the complexity of polynomial expressions by evolutionary operations.

Let $\{q_1, \ldots, q_k\}$ be a set of monomials over the variables $X_1, \ldots, X_m$. The encoding of that set using d bits per exponent is a binary list such that

1. the initial segment of k bits defines the active state of each monomial;
2. the remaining bits are split into k segments of size $m \times d$, each representing a monomial;
3. the bits in each monomial segment are split into m sub-segments of size d. The j^{th} sub-segment is the binary representation of the degree of the variable X_j in the enclosing monomial segment;

This encoding can also be viewed as the flattening of the binary exponents in the matrix representation prefixed by the activation segment. The set $\{x_1^3x_3, x_3^7, x_1x_2\}$

(with $m = 3$) has matrix representation

$$\begin{bmatrix} 3 & 0 & 1 \\ 0 & 0 & 7 \\ 1 & 1 & 0 \\ 6 & 2 & 5 \end{bmatrix} = \begin{bmatrix} 011 & 000 & 001 \\ 000 & 000 & 111 \\ 001 & 001 & 000 \\ 110 & 010 & 101 \end{bmatrix}_{(2)}$$

where the right matrix is in binary form using $d = 3$ bits. An example of this set of monomials with the forth monomial, $x_1^6 x_2^2 x_3^5$, inactive would be

1110; 011, 000, 001; 000,000,111; 001,001,000; 110,010,101

where, for reading purposes, semicolons separate segments and commas separate variables. The first $k = 4$ bits inform that the first, second and third monomials are active while the fourth is not.

While each valid encoding represents a set of monomials the map is not bijective: each set of monomials has multiple encodings, for example by changing d or the order of monomial segments. However, considering the EPR task, this is a minor issue and a bijective map would add computational complexity and negative impact to the algorithm's performance — The encoding example should include the inactive term so that both examples are consistent (lines 113-115). Is there a reference some experimentation to back this up? We changed the text according to the suggestion..

There is one final remark concerning this encoding method. As it is, the activation segment can become all zeros, representing the empty set of monomials. This situation can be avoided with a simple hack: Given an encoding, the first monomial is always considered active, thus restricting the syntactic form of encodings to binary strings starting with 1. In practice, this means that the implementation of the encoding can omit the first bit.

2.2. Cost Function

The polynomial regression error considered so far accounts for the ability to predict the transformed testset. A known problem of using a cost function

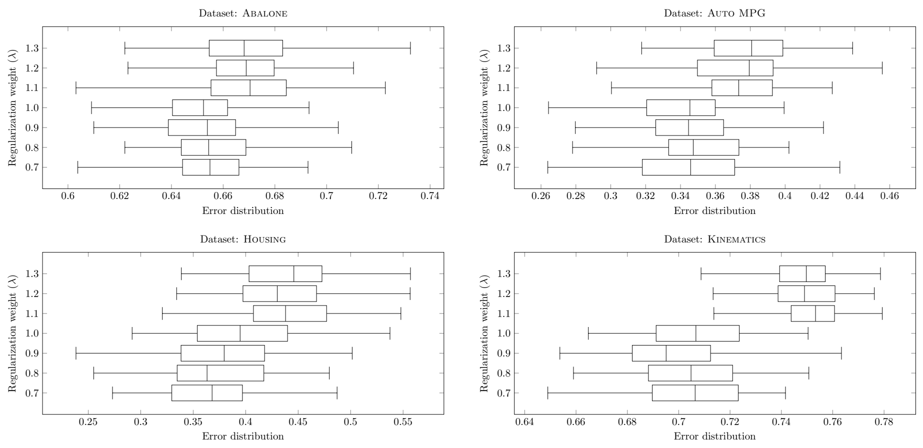

Figure 1: Error distribution by regularization exponent for common datasets. The box plots summarize error values of 75 simulations (60 iterations on a population size of 50) for each value of λ. Performance of the non-regularized EPR is plotted in the line $\lambda = 1$. Performance for $\lambda < 1$ is better than $\lambda > 1$ in all datasets.

based only in the dataset error (and of polynomial regressions in general) is the tendency to overfit training data. Excessive variance of the estimation method can be reduced by regularizing the error function with a penalty factor. Thus, to reduce polynomial complexity and variance by regularizing the size of the monomial set the error function from equation 2 is multiplied by a factor λ^k

$$J_{\text{reg}}(\Theta, \lambda; \mathcal{Q}, \mathcal{D}) \quad = \quad \lambda^k J_{\text{fit}}(\Theta; \mathcal{Q}, \mathcal{D}) \tag{4}$$

where k is the number of monomials in the polynomial. When $\lambda > 1$ polynomials with more monomials are penalized. The regularized extension of EPR is denoted by Evolutionary Polynomial Regression with Regularization (EPRR).

A simple exploration on the effect of the regularization parameter is depicted in Figure 1 where it is possible to observe that penalizing polynomial complexity (i.e. $\lambda < 1$) achieves better results than the opposite ($\lambda > 1$). This observation motivates further inquire, done in section 3 where we compare the regularized versions with $\lambda < 1$ and the non-regularized algorithm against several other regression methods. — please clarify this paragraph and justify this claim we rephrased the statement.

2.3. Genetic Algorithm Parameterization

— Please indicate all these values in detail We described the parameterization in more detail.

— You should contrast obtained results in a table. Results are hard to judge from Fig 3 alone. Done that.

— The number of 250 for iterations seems premature specially when one looks at Fig 4. Stabilize? Really? This only seems valid in one case of lambda. we changed the number of iterations to 2000, way behind stabilization

In general GAs offer many possibilities with respect to the choice of genetic operators and respective application rates, population evolution, *etc.* The results found here where obtained using the package **genalg** [?] with standard operators (crossover and mutation) and population evolution defined by mutation rate of 5% and 20% elitism between generations.

3. Experimental Results

Here is described the experiment setup used to gather and summarize the empirical evidence that supports this comparative study of EPR and EPRR. Evaluation is focused in error distribution and, besides EPR and EPRR, also uses several common regression methods and datasets easily accessible in R, the free software environment for statistical computing and graphics [?][2]. A small consideration on the convergence speed concludes this section.

3.1. Regression Methods and Datasets

The EPRR method is ranked against several well-known learning algorithms for regression, namely: non-regularized EPR, Linear Regression, Support Vector

[2]The datasets and R code used to produce the results and plots in this paper are available online at `https://github.com/jpneto/GenAlgPoly`.

dataset	**method**	**quantile 25%**	**error mean**	**quantile 75%**
Abalone	EPRR $\lambda = 0.7$	0.6392	0.6555	0.6677
	EPRR $\lambda = 0.8$	0.6408	0.6543	0.6636
	EPRR $\lambda = 0.9$	0.6481	0.6581	0.6707
	EPRR $\lambda = 1.0$	0.6542	0.6715	0.6816
	Linear Regression	0.6803	0.6927	0.7078
	SVM (linear kernel)	0.6916	0.7044	0.7205
	Regression Trees	0.7423	0.7520	0.7621
	Random Forest	0.6585	0.6695	0.6814
	Cond. Inference Trees	0.7031	0.7126	0.7264
Auto-Mpg	EPRR $\lambda = 0.7$	0.3635	0.3916	0.4147
	EPRR $\lambda = 0.8$	0.3629	0.3956	0.4228
	EPRR $\lambda = 0.9$	0.3646	0.4130	0.4215
	EPRR $\lambda = 1.0$	0.3691	0.3999	0.4057
	Linear Regression	0.4071	0.4284	0.4473
	SVM (linear kernel)	0.4116	0.4358	0.4613
	Regression Trees	0.4216	0.4501	0.4785
	Random Forest	0.3318	0.3624	0.3892
	Cond. Inference Trees	0.4063	0.4372	0.4663
Housing	EPRR $\lambda = 0.7$	0.4412	0.6650	0.5739
	EPRR $\lambda = 0.8$	0.4241	0.5274	0.6016
	EPRR $\lambda = 0.9$	0.4354	0.5717	0.6462
	EPRR $\lambda = 1.0$	0.4477	0.5417	0.5995
	Linear Regression	0.4898	0.5313	0.5649
	SVM (linear kernel)	0.4831	0.5469	0.6017
	Regression Trees	0.4845	0.5232	0.5720
	Random Forest	0.3283	0.3679	0.4023
	Cond. Inference Trees	0.4676	0.5080	0.5413
Kinematics	EPRR $\lambda = 0.7$	0.6600	0.6660	0.6720
	EPRR $\lambda = 0.8$	0.6617	0.6694	0.6751
	EPRR $\lambda = 0.9$	0.6636	0.6714	0.6761
	EPRR $\lambda = 1.0$	0.7568	0.7558	0.7739
	Linear Regression	0.7672	0.7759	0.7849
	SVM (linear kernel)	0.8074	0.8136	0.8247
	Regression Trees	0.5673	0.6021	0.5803
	Random Forest	0.7386	0.7344	0.7645
	Cond. Inference Trees	0.7558	0.7620	0.7686

Table 1: Tabular summary results for different regression methods on common datasets. Although EPRR not always achieves the smallest expected error, performance is on-par with more sophisticated methods.

Machines [**?**] with linear kernel, Regression Trees [**?**], Random Forest [**?** **?**] and Conditional Inference Trees [**?**].

The performance of each method is evaluated on several common datasets. From each dataset 70% of the observations are reserved for training purposes and the remaining observations used to estimate the error. To enhance the robustness of results this process is repeated 75 times, each time with a different shuffling of the samples in the train and test sets. The datasets have been scaled to prevent problems with different magnitude attributes. The box plots in figures 2 and 3 resume the test set error distributions over these different runs.

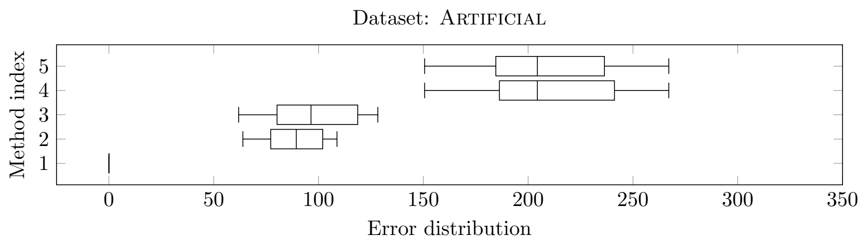

Figure 2: Testing polynomial discovery. The dataset is generated from a polynomial expression and, as shown, EPRR finds the exact generator structure: in line 1, the error box is centered in 0 and has width 0. The regression methods depicted are: 1. EPRR, 2. Linear Regression, 3. SVM, 4. Regression Trees and 5. Conditional Inference Trees

One of the used datasets, Artificial, has a special role: it is used to test if EPRR is able to discover a polynomial model. The idea of this test is to generate a polynomial dependent variable and measure the EPRR error after fitting the dataset. The genetic algorithm parameterization for this dataset uses a population with size $n = 100$ and evolves for 50 generations. For the remaining datasets the population has size $n = 300$ and evolves for 100 generations.

Artificial is a polynomial dataset with four numeric features, $x_1, \ldots x_4$, where x_1, x_3 are outcomes from Poisson random variables, and x_2, x_4 from Normal random variables. The dependent variable is given by the polynomial expression $y = x_2 x_4^2 + x_1^2 x_3 + 5$. The dataset includes $n = 50$ observations;

Housing concerns the task of predicting housing values in areas of Boston. There are $n = 506$ observations of $m = 13$ continuous attributes and one dependent variable, the median value of owner-occupied homes in thousands of USD;

Abalone is used to predict the age of a abalone shell using $m = 8$ numeric attributes concerning several physical measurements. There are $n = 4177$ observations;

Auto MPG gathers fuel consumption in miles per gallon, based on two dis-

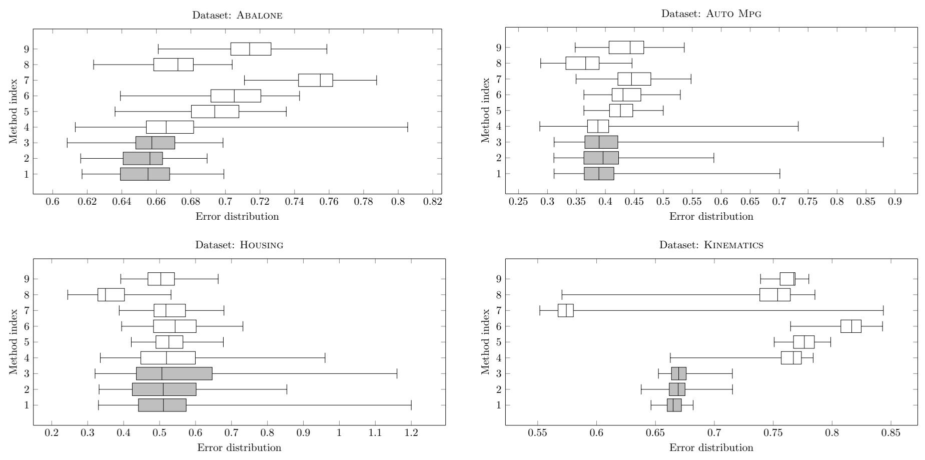

Figure 3: Graphical summary results for different regression methods on common datasets. Although EPRR not always achieves the smallest expected error, performance is on-par with more sophisticated methods. The regression methods depicted in these figures are: 1. EPRR, $\lambda = 0.7$; 2. EPRR, $\lambda = 0.8$; 3. EPRR, $\lambda = 0.9$; 4. EPR (i.e. $\lambda = 1.0$); 5. Linear Regression; 6. SVM (linear kernel); 7. Regression Trees; 8. Random Forest (with 100 trees); 9. Conditional Inference Trees.

crete and five continuous attributes ($m = 7$). There are $n = 398$ observations;

Kinematics results from a realistic simulation of the forward kinematics of an 8 link robot arm. The task is to predict the distance of the end-effector from a target using $m = 8$ continuous attributes. There are $n = 8192$ observations;

3.2. Convergence speed

Since this work is oriented to the EPRR model error it is necessary to assess how this depends on the number of generations of the GA. As illustrated in Figure 4, the error quickly drops during the initial 100 generations. Then, it proceeds slower achieving better solutions only with marginal error reduction. The EPR takes more than 300 generations to achieve similar error results.

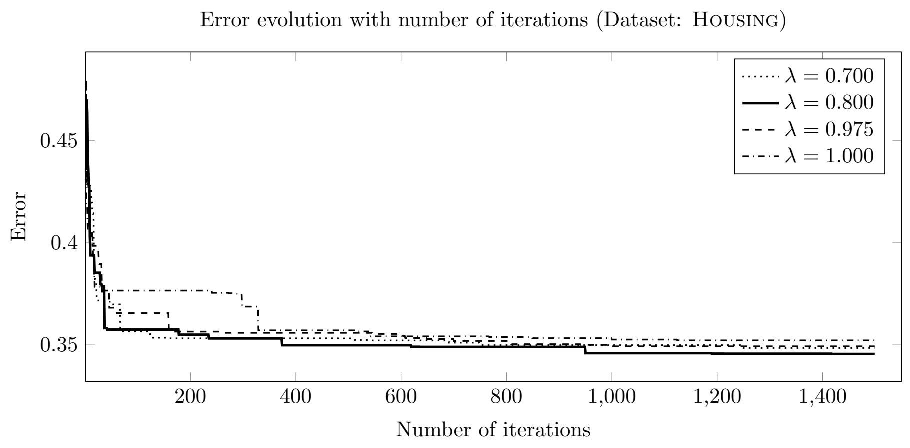

Figure 4: Learning curve: Error progress for the HOUSING dataset during a single execution of the genetic algorithm. The figure shows the fitness evolution for different regularization values. The population of each run consists of 200 polynomials.

4. Conclusion and Future Work

Of the regression methods considered Random Forests (RF) achieves the best results in two out of four datasets. If we compare against non-ensemble methods, EPR and EPRR outperform the other estimators, with the exception of Regression Trees in the Kinematics dataset.

Comparing EPR and EPRR — the main article's topic — the regularized version achieves better results at ABALONE and especially KINEMATICS. A Bayesian Estimation was computed, which estimates the difference in means between two groups and yields a probability distribution over the difference. On the Abalone dataset errors are improved wrt EPR in a difference in means, for all tested λ values less than 1, resulted in a 95% HDI (Highest Density Interval) equal to $[0.01, 0.02]$ which achieves statistical significance. On the Auto-MPG dataset the 95% HDI is $[-0.01, 0.008]$ which includes zero. On the Housing dataset the 95% HDI is $[-0.02, 0.05]$, for $\lambda = 0.7$, which includes zero (the other values of λ had similar statistics).

For complexity considerations EPR and EPRR demand some processing time. On a quad-core computer, processing the Kinematics dataset (with near $8K$ observations) takes approximately 5 minutes. Probably processing time can be reduced by one to two orders in magnitude if the algorithm is implemented with computational speed in mind. However, speed optimization is not the focus of this article.

Parameters like mutation chance or the amount of elitism can be tested for tuning purposes. However, these type of tests need a low-level, fast implementation of EPR and are postponed to future investigation.

Acknowledgements

The authors are grateful to the Fundação para a Ciência e Tecnologia (FCT) and the R&D laboratory LabMAg for the financial support given to this work, under the strategic project PEst-OE/EEI/UI0434/2011.

João Pedro Neto's work is supported by centre grant (to BioISI, Centre Reference: UID/MULTI/04046/2013), from FCT/MCTES/PIDDAC, Portugal.

Datasets used herein are selected from Luís Torgo's data repository, `http://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html`. Most can also be found in the UCI ML repository at `http://archive.ics.uci.edu/ml/`.

The authors wish to thank professor André Falcão for motivation and useful discussions around the article.