Selection of Polynomial Features by Genetic Algorithms

Francisco Coelho João Neto

[DRAFT July 22, 2013]

Abstract

Many applications require models that have no acceptable linear approximation and many nonlinear models are defined by polynomials. The use of genetic algorithms to find polynomial models is decades old but still poses challenges due to the complexity of the search and different definitions of optimal solution.

This paper describes a two-step method that uses genetic algorithms and linear regression to find empirical polynomial regressions. Experiments on common datasets show that, discounting the training computational effort, this method is quite competitive.

1 Introduction

With notable exceptions (*e.g.* neural networks) machine learning regression techniques are based on 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 ecology, cybernetics, robotics and other areas.

Polynomials, one of the most studied subjects in mathematics, generalize linear functions and define, perhaps, the simplest and most used nonlinear models. Applications include colorimetric calibration [17], explicit formulae for turbulent pipe flows [8], computational linguistics [20] and more recently, analytical techniques for cultural heritage materials [7], liquid epoxy molding process [5], B-spline surface reconstruction [9], product design [6] or forecasting cyanotoxins presence in water reservoirs [10]. This not only illustrates the wide spectrum of applications but, additionally, work in each one of these polynomial models uses, at some point, a genetic algorithm.

Genetic algorithms (GA) where, 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 locally optimal parameters. Research related to genetic algorithms abound and spans from the 1950s seminal work of Nils Aall Barricelli [1] 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 come from John Holland's work and his book "Adaptation in Natural and Artificial Systems" [12].

One interesting take on genetic algorithms, named *genetic programming* by John Koza [14], proposed the use of GAs to search the syntactic structure of complex functions. This syntatic structure search is keen to the central ideas of deep learning [3, 2], a subarea of machine learning actually producing quite promising results (*e.g.* [23]). 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 [16, 26, 25] but still generates original research [11, 4]. In line with that research this work describes a general method to find a polynomial regression of a given dataset. The optimal regression minimises a cost function that accounts for both the root-mean-square error (error) and a regularization factor to avoid over-fitting.

It turns out that, discarding the computational cost of training, the polynomial regression method presented here, Genetic Algorithms for Polynomials (GAPoly), provides a quite competitive regression method. Indeed, it is only systematically out-performed by random forests, an *ensemble* method.

The remainder of this paper is organized as usual: the next section describes the details of our method 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 is dedicated to the description of an algorithm to find a polynomial regression from a given dataset. It starts with a brief introduction and outline of the algorithm 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

$$p\left(x_1, \ldots, x_k\right) = \sum_i \theta_i m_i$$

where each m_i is a monomial, $m_i = \prod_j x_j^{\alpha_{ij}}$, the exponents are non-negative integers, $\alpha_{ij} \in \mathbb{N}_0$, and the coefficients are real valued, $\theta_i \in \mathbb{R}$. 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 $m_1 = x_1, m_2 = x_2x_3$ and $m_3 = x_1^2x_3$, coefficients $\theta_1 = 2, \theta_2 = 1$ and $\theta_3 = 1/2$ and exponents $\alpha_{1,1} = 1, \alpha_{2,2} = 1, \alpha_{2,3} = 1, \alpha_{3,1} = 2, \alpha_{3,3} = 1$ and all other $\alpha_{ij} = 0$. The exponents alone are a matrix that defines the monomial structure of the polynomial, $A = [\alpha_{ij}]$. For the example above

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 2 & 0 & 1 \end{bmatrix} \sim \begin{bmatrix} x_1 \\ x_2x_3 \\ x_1^2x_3 \end{bmatrix}$$

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 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 exponentialy with either n or d. But more importantly, the polynomial regression problem can be naturally split into two subproblems:

1. for a given set of monomials $m_1, \ldots, m_R$, find the regression coefficients $\theta_1, \ldots, \theta_R$ 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 $Y, X_1, \ldots, X_n$ and $\mathcal{M} = \{m_1, \ldots, m_R\}$ a set of R monomial expressions over $X_1, \ldots, X_n$. Define the hypothesis[1]

$$h_{\Theta,\mathcal{M}}\left(x_1, \ldots, x_n\right) \;=\; \sum_{j=1}^{R} \theta_j m_j|_{X_i = x_i, \forall 1 \leq i \leq n}$$

and let the cost

$$J_{fit}\left(\Theta; \mathcal{M}, \mathcal{D}\right) \;=\; \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} - h_{\Theta,\mathcal{M}}\left(x_1^{(i)}, \ldots, x_n^{(i)}\right)\right)^2} \tag{1}$$

[1]The expression $m|_{X=x}$ reads "*replace all instances of* X *by* x *in* m".

Algorithm 1 GAPoly uses linear regression to find monomial coefficients that minimize the error over a dataset and GAs to explore the space of polynomials. At exit the error of the fittest instance is bounded by ϵ.

```
function GAPoly(D, pop_0, ε)
    pop ← pop_0; err ← 1.0 + ε
    while err > ε do
        pop ← IterateGA(pop)                          ▷ Build next generation
        pop ← Sort(pop, key = Lm.rmse)    ▷ Sort by error of linear regression
        err ← rmse (First(pop))
    end while
    return First(pop)
end function
```

be the usual root-mean-square error (error) function. Now the first problem can be stated as:

> Given a dataset $\mathcal{D}$ and a set of monomials $\mathcal{M}$, find parameters Θ that minimize $J(\Theta; \mathcal{M}, \mathcal{D})$.

It turns out that this problem can be solved as a usual linear regression problem by expanding $\mathcal{D}$ with columns that replicate the monomials in $\mathcal{M}$.

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 $\mathcal{M}_p$ be the set of monomials in p (without the coefficients) and compute the fitness

$$\phi_p \quad = \quad \min_{\Theta} J(\Theta; \mathcal{M}_p, \mathcal{D})$$

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

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

2.1 Polinomial Encoding

The encoding for any polynomial will be as follows:

1. an initial segment detailing which monomials are active (the 1st monomial is always active), this is represented in unary description, i.e., each monomial is identified by a single bit

2. the remaining is split into m sets of bits of equal size, each one representing a monomial

3. each monomial is split into n sets of d size each, i.e., a variable

4. for each variable, the remaining bits give the binary description of the variable degree, i.e., the maximum exponent is given by $2^d - 1$

Let's see an example: consider polynomial $x_1^3 x_3 + x_3^7 + x_1 x_2$ with $m = 4, n = 3$ and $d = 3$. One possible encoding would be:

$$110 - 011, 000, 001 \quad ; \quad 000, 000, 111 \quad ; \quad 001, 001, 000 \quad ; \quad 110, 010, 101$$

(for reading purposes the semicolons separate monomials, the commas separate variables)

The first three bits inform that the second and third monomials are active while the fourth is not (as said, the first monomial is always active). This last monomial does not enter neither in the polynomial regression nor in the fitness evaluation. However, it acts as a kind of junk DNA, becoming active when, in a future mutation or crossover, the third bit of the entire sequence flips from 0 to 1.

Let's interpret the first monomial description, $011, 000, 001$. It is divided by three since $n = 3$. The first triple 011 is the binary description of the exponent of variable x_1 which is 3, so the first monomial includes x_1^3. The second triple, 000, means that x_2 is not part of the monomial. The third triple 001 says that variable x_3 has exponent 1, so the first monomial consists of $x_1^3 x_3$. All the remaining sets of nine bits are interpreted the same way and we get the previous polynomial.

Notice that all binary descriptions give rise to valid polynomials. Notice however that this is not a bijective mapping. For each polynomial there are multiple representations. For example, $x_1 + x_2$ and $x_2 + x_1$ have different representations. The authors considered that more complex mappings in order to force a one to one mapping would impact negatively in the algorithm performance without giving anything in return.

If the coding consists of entirely zeros, by convention, it describes polynomial x_1. This has to do with the execution of the polynomial regression that would fail if we interpret it as the zero polynomial. Anyway, for progressive larger binary descriptions, the chances of getting this zero description decrease exponentially, so it does not impact in any meaningful way in the algorithm's performance.

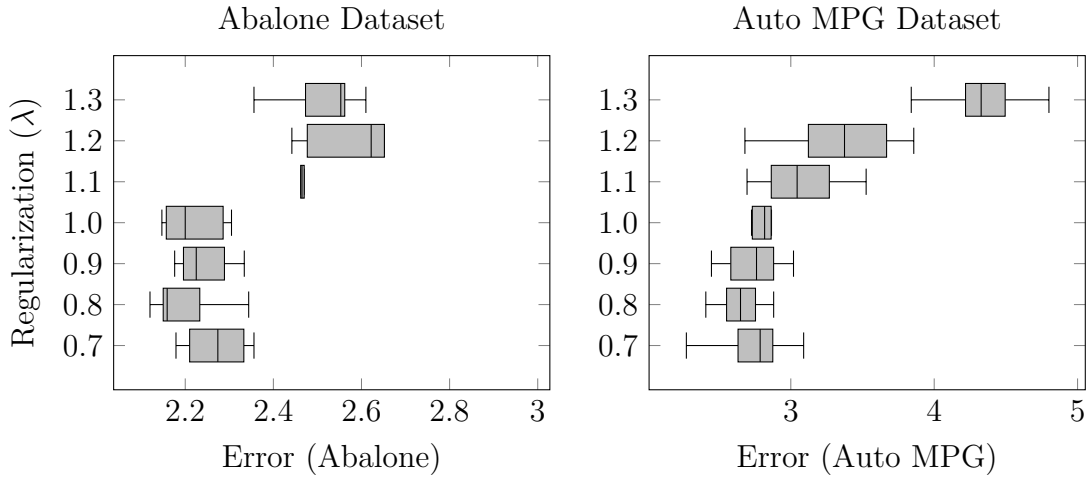

Figure 1: Error distribution by regularisation exponent for the Abalone dataset.

2.2 Cost Function

The polynomial fitness considered so far is based on the ability to predict the test set after the polynomial regression has found the appropriate coefficients θ_i for each one of the polynomial's monomials m_i.

This fitness function tends to prefer more complex polynomials, namely in the number of monomials which provides the regression algorithm for more fitting possibilities. One way to balance this is to provide a regularization term into the fitness function. Our proposal is to include a multiplicative factor into the fitness function proportional to the number of monomials. Thus, the fitness function from equation 1 becomes

$$J_{reg}\left(\Theta; \mathcal{M}, \mathcal{D}\right) \quad = \quad \lambda^m J_{fit}\left(\Theta; \mathcal{M}, \mathcal{D}\right) \tag{2}$$

where m is the number of monomials the polynomial has. A λ greater than one punishes polynomials with more monomials.

Somewhat unexpectedly after some experiences it was found that lower values for λ sometimes provide better, even if marginal, results. Figure 1 shows results for the Abalone dataset with ten runs for each λ, and figure **??** presents regularization results for the Auto-MPG dataset. The following section Experimental Results includes information about these datasets.

The typical inflection point lies around $\lambda = 0.8$. The following results use the regularisation parameter with this value.

2.3 Genetic Operators

It was used the R package genalg [27] for the execution of genetic algorithm. The operators were the standard ones: (a) crossover, i.e., a pair of solutions from the previous generations are combined by splitting and mixing their respective representations, and (b) mutation, changing the values of single bits; the mutation chance applied in the datasets was 5%. There was also elitism between generations, i.e., 20% of the best solutions are kept in the next generation.

3 Experimental Results

- Measured quantities
  - error
  - number of iterations to convergence
  - memory usage
  - F1, ROC, ?
- Selection of datasets and regression algorithms
- Summary Figures and Numeric results

These results were found using R's programming language [19].

To compare this paper's proposed algorithm we applied the exact same train and test samples using other learning algorithms, namely: Linear Regression, Support Vector Machines [18], Regression Trees [24], Conditional Inference Trees [13, 22, 21] and Random Forests [15]

In order to train and test the performance of GAPoly it was used several datasets (see below). For each dataset, we selected 70% for training purposes and the remaining observations to make a test set in order to compute the estimated error. To achieve more robust results, each dataset were processed 25 times, each one with different samples for the train and test sets. For the datasets with attribute values of different magnitudes, a preliminary scaling was executed. The results below are boxplots for the test set error predictions over these different runs.

Artificial: this is an artificial 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 y is given by expression $x_2 x_4^2 + x_1^2 x_3 + 5$. The dataset includes 50 observations and, only in this case, there were executed just 10 runs.

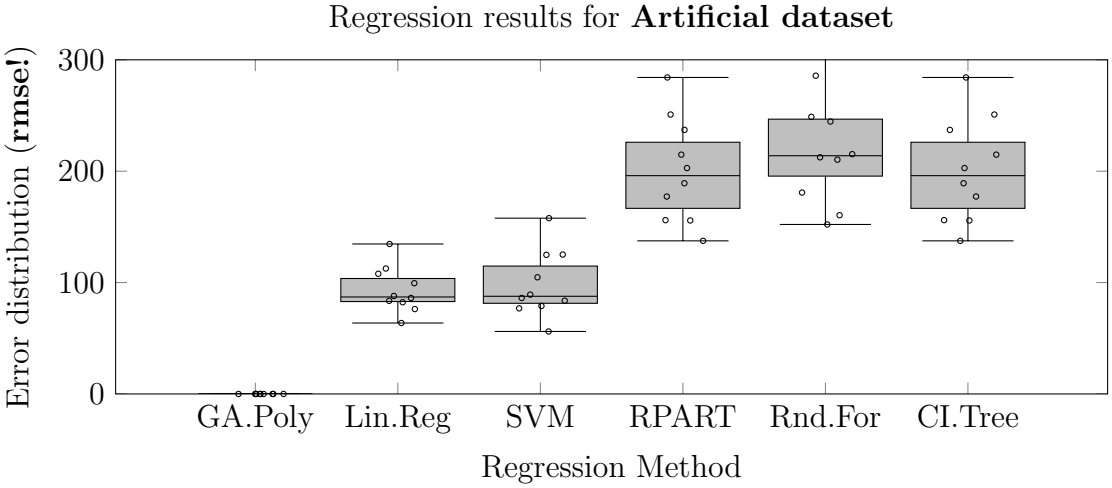

Figure 2: results for Artificial dataset

This dataset was used in order to verify if GAPoly was able to find the polynomial relation, which the algorithm did (cf. figure 2). In this case, we used a population of 100 solutions with 50 iterations for each run.

In the next datasets, the population has size 250 with 100 iterations for each run.

Housing: This data set concerns the task of predicting housing values in areas of Boston. There are 13 continuous attributes and the dependent variable is the median value of owner-occupied homes in \$1000's[2]. There are 506 observations. Figure 3 presents the results for this dataset.

Just as an example of the model the GAPoly algorithm outputs: in this dataset the best polynomial was the following:

$$y = -0.12x_6x_9^4 + 0.78x_6 - 0.4x_9^2x_{13} + 0.17x_{13}^2 - 0.044$$

where the attributes mean: x_6, RM average number of rooms per dwelling; x_9, RAD index of accessibility to radial highways; x_{13}, Lower status of the population.

For comparisation if we access the mean decrease in accuracy found by Random Forests, the most important attributes are – in decreasing order – x_{13}, x_6, x_5 (but x_5 is already considered 4 times less important than x_6). Both algorithms agree in two of their three most important attributes.

[2]This and the remaining regression datasets were selected from Luis Torgo's data repository, http://www.dcc.fc.up.pt/ ltorgo/Regression/DataSets.html. Most of these datasets originally come from UCI ML repository, http://archive.ics.uci.edu/ml/

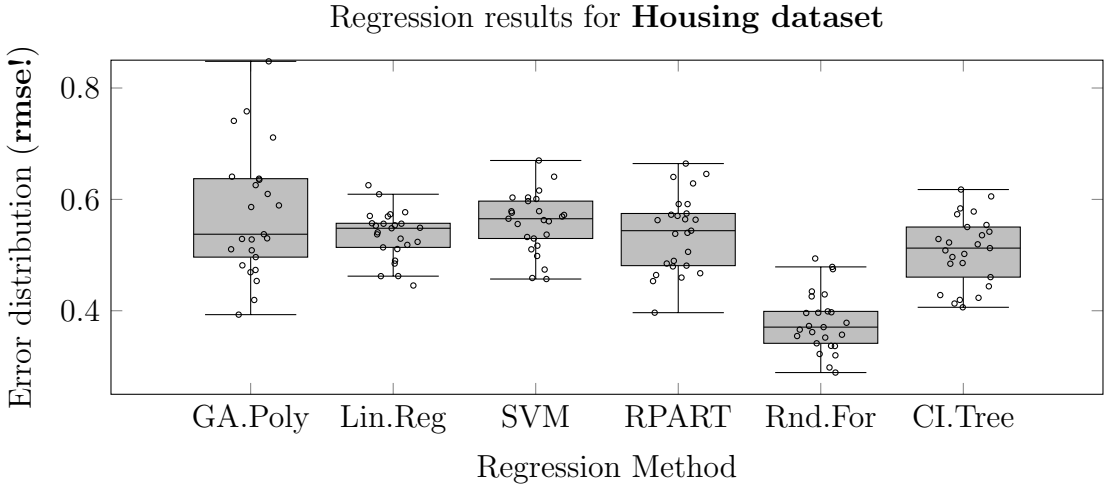

Figure 3: results for Boston Housing dataset

Abalone: This dataset can be used to predict the age of a abalone shell using the given 8 numeric attributes concerning several physical measurements. There are 4177 observations. Figure 4 presents the results for this dataset.

Auto-MPG: This dataset is used to predict fuel consumption in miles per gallon, based on two discrete and five continuous attributes. There are 398 observations. Figure 5 presents the results for this dataset.

Kinematics: This dataset is concerned with the 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 8 continuous attributes. There are 8192 observations. Figure 6 presents the results for this dataset.

3.1 Convergence speed

The GA quickly proceeds in the first 50 to 100 generations to reasonable error rates. Then, it proceeds slower achieving best solutions with marginal error reduction. Since the entire learning process takes some time, in the current R implementation[3], placing a limit between 50 to 100 generations already achieves good results, relative to higher iteration values. Figure 7 shows a typical error evolution for the dataset Abalone given two different values for λ.

[3]The processing time can probably be speeded by one to two orders in magnitude if the process is implemented in a low level programming language like C++. However, speed optimisation was not the focus of this article

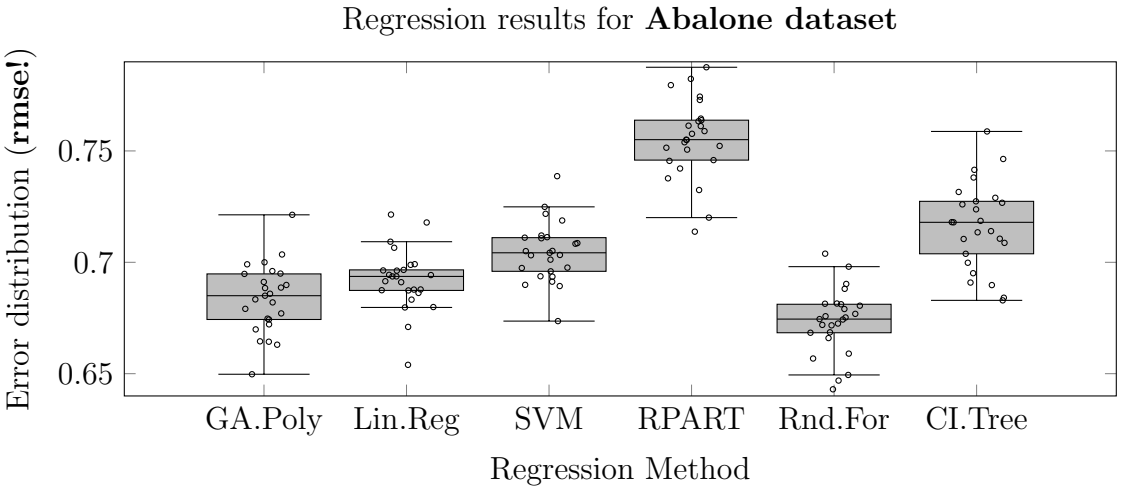

Figure 4: results for Abalone dataset

4 Conclusion

The proposed method has competitive results comparing with some well-known regression methods. Only Random Forest outperforms GAPoly systematically (it also outperforms all the other regression algorithms). One exception – the artificial dataset that uses a straightforward polynomial relation – is the Auto-MPG datatest where GAPoly has comparable results. This is evidence that applying standard genetic operators for polynomial model searching is a viable tool for regression purposes.

5 Bibliography

References

[1] Nils Aall Barricelli. Numerical testing of evolution theories. part i: Theoretical introduction and basic tests. *Acta Biotheoretica*, 16(1-2):69–98, 1962.

[2] Yoshua Bengio. Learning deep architectures for AI. *Foundations and trends in Machine Learning*, 2(1):1–127, 2009.

[3] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. 2013.

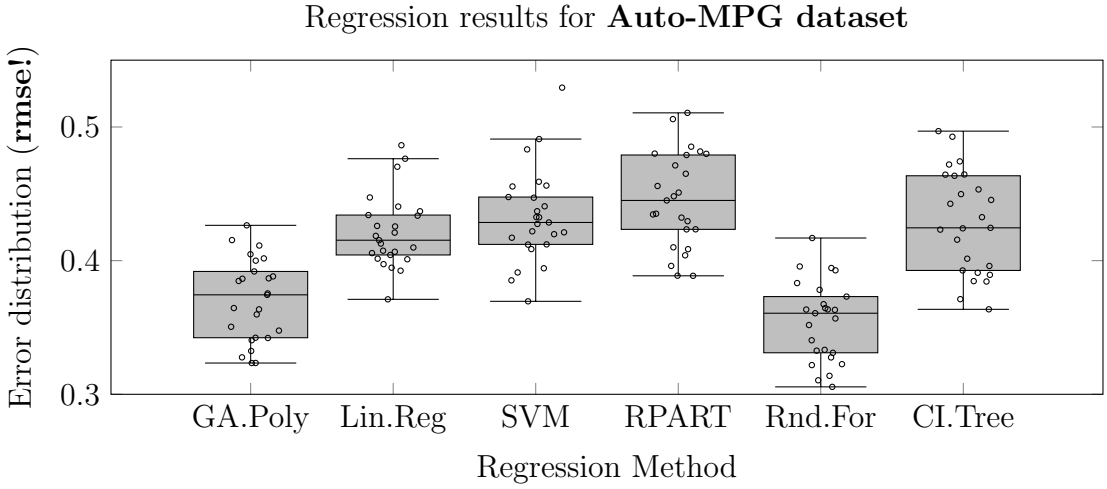

Figure 5: results for Auto-MPG dataset

[4] B Cetisli and H Kalkan. Polynomial curve fitting with varying real powers. *Electronics and Electrical Engineering*, 112(6):117–122, 2011.

[5] Kit Yan Chan, Tharam S Dillon, and Che Kit Kwong. Modeling of a liquid epoxy molding process using a particle swarm optimization-based fuzzy regression approach. *Industrial Informatics, IEEE Transactions on*, 7(1):148–158, 2011.

[6] Kit Yan Chan, CK Kwong, and Tharam S Dillon. Development of product design models using fuzzy regression based genetic programming. In *Computational Intelligence Techniques for New Product Design*, pages 111–128. Springer, 2012.

[7] L Cséfalvayová, M Pelikan, I Kralj Cigić, J Kolar, and M Strlič. Use of genetic algorithms with multivariate regression for determination of gelatine in historic papers based on FT-IR and NIR spectral data. *Talanta*, 82(5):1784–1790, 2010.

[8] J Davidson, D Savic, and G Walters. Method for the identification of explicit polynomial formulae for the friction in turbulent pipe flow. *Journal of Hydroinformatics*, 1:115–126, 1999.

[9] Akemi Gálvez, Andrés Iglesias, and Jaime Puig-Pey. Iterative two-step genetic-algorithm-based method for efficient polynomial b-spline surface reconstruction. *Information Sciences*, 182(1):56–76, 2012.

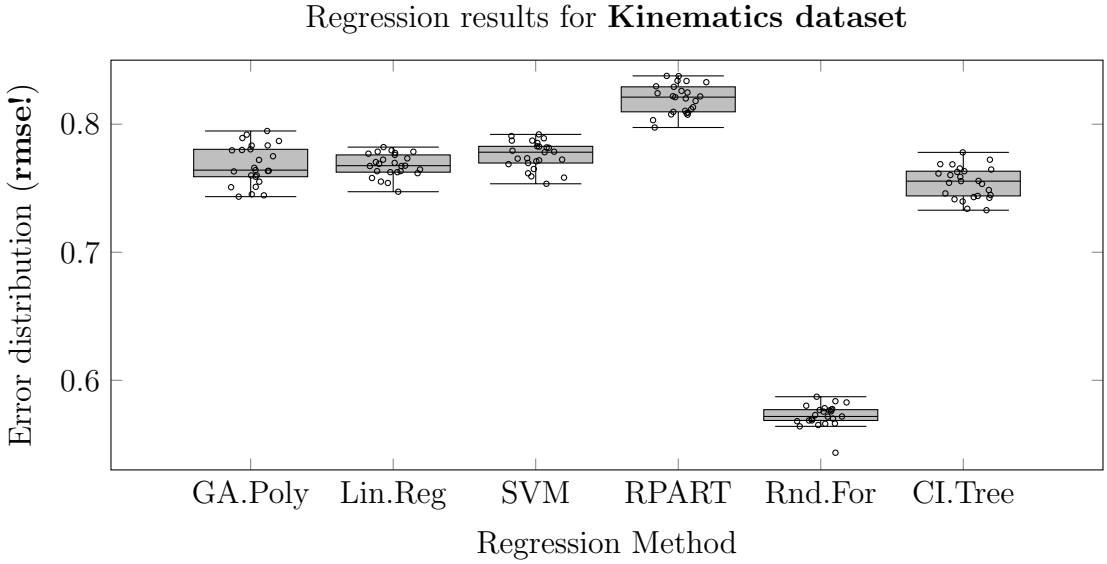

Figure 6: results for Kinematics dataset

[10] Paulino José García Nieto, JR Alonso Fernández, FJ de Cos Juez, Fernando Sánchez Lasheras, and C Díaz Muñiz. Hybrid modelling based on support vector regression with genetic algorithms in forecasting the cyanotoxins presence in the trasona reservoir (northern spain). *Environmental research*, 2013.

[11] Magnus Hofwing, Niclas Strömberg, and Martin Tapankov. Optimal polynomial regression models by using a genetic algorithm. In *Proceedings of the Second International Conference on Soft ComputingTechnology in Civil, Structural and Environmental Engineering Conference,(Crete, Greece), 2011009*, 2011.

[12] John H Holland. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975.

[13] Torsten Hothorn, Kurt Hornik, and Achim Zeileis. Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical Statistics*, 15(3):651–674, 2006.

[14] John R Koza. *Genetic Programming: vol. 1, On the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

[15] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002.

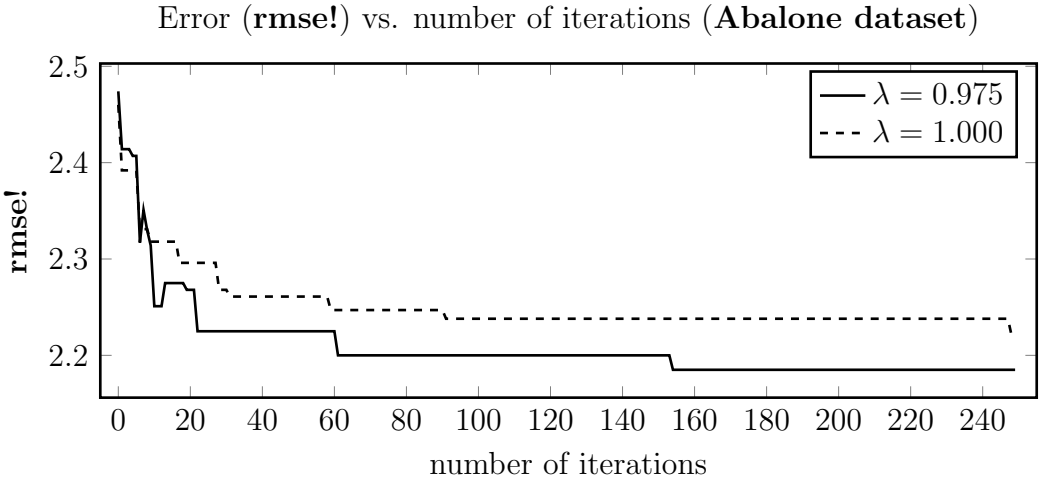

Figure 7: fitness progress for Abalone dataset (population 200)

[16] Koen Maertens, Josse De Baerdemaeker, and R Babuška. Genetic polynomial regression as input selection algorithm for non-linear identification. *Soft Computing*, 10(9):785–795, 2006.

[17] L Mendes and P d Carvalho. Adaptive polynomial regression for colorimetric scanner calibration using genetic algorithms. In *Intelligent Signal Processing, 2005 IEEE International Workshop on*, pages 22–27. IEEE, 2005.

[18] David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, and Friedrich Leisch. e1071: Misc functions of the department of statistics (e1071), tu wien. 2012. R package version 1.6-1.

[19] R Core Team. R: A language and environment for statistical computing. 2013.

[20] Luciano Sánchez, José Otero, and Inés Couso. Obtaining linguistic fuzzy rule-based regression models from imprecise data with multiobjective genetic algorithms. *Soft Computing*, 13(5):467–479, 2009.

[21] Carolin Strobl, Anne-Laure Boulesteix, Thomas Kneib, Thomas Augustin, and Achim Zeileis. Conditional variable importance for random forests. *BMC Bioinformatics*, 9(307), 2008.

[22] Carolin Strobl, Anne-Laure Boulesteix, Achim Zeileis, and Torsten Hothorn. Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC Bioinformatics*, 8(25), 2007.

[23] Daniel Tarlow, Ilya Sutskever, and Richard S Zemel. Stochastic k-neighborhood selection for supervised and unsupervised learning.

[24] Terry Therneau, Beth Atkinson, and Brian Ripley. rpart: Recursive partitioning. 2013. R package version 4.1-1.

[25] Chih-Hung Wu, Gwo-Hshiung Tzeng, and Rong-Ho Lin. A novel hybrid genetic algorithm for kernel function and parameter optimization in support vector regression. *Expert Systems with Applications*, 36(3):4725–4735, 2009.

[26] Tian-Li Yu and Wei-Kai Lin. Optimal sampling of genetic algorithms on polynomial regression. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1089–1096. ACM, 2008.

[27] Egon Willighagen. genalg: R Based Genetic Algorithm 2012 R package version 0.1.1