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 [?], explicit formulae for turbulent pipe flows [?], computational linguistics [?] and more recently, analytical techniques for cultural heritage materials [?], liquid epoxy molding process [?], B-spline surface reconstruction [?], product design [?] or forecasting cyanotoxins presence in water reservoirs [?]. 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 [?] 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" [?].

One interesting take on genetic algorithms, named *genetic programming* by John Koza [?], 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 [?, ?], a subarea of machine learning actually producing quite promising results (*e.g.* [?]). 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 [?, ?]. 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 $p(x_1, x_2, x_3) = 2x_1 + x_2x_3 + \frac{1}{2}x_1^2x_3$ 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]

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

and let the cost

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

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

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

> 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

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

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.

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.

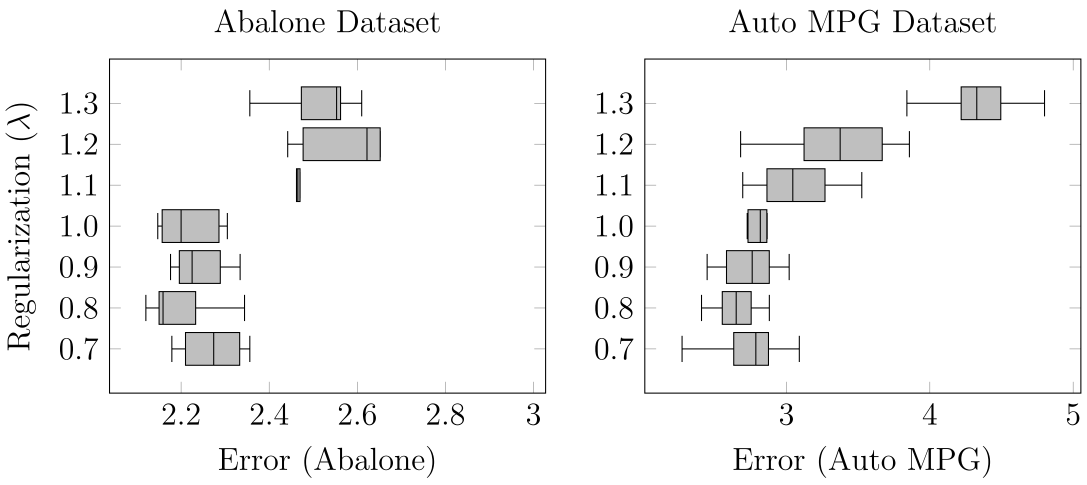

Figure 1: Error distribution by regularisation exponent for the Abalone dataset.

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

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

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.