0.1 Polynomial Encoding

The chromosome coding will as follow:

1. an initial segment detailing which monomials are active (the 1st monomial is always active), these are represented in unary description
2. the chromosome is split into 'max.monomials' sets of bits of equal size (a monomial)
3. each monomial is split into 'n.vars' sets of 'max.degree' 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 $2^m ax.degree-1$

Eg: consider polynomial $x_1^3x_3+x_3^7+x_1x_2$ with max.monomials = 4, n.vars = 3 and max.degree=3. One possible encoding would be:

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

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

The first 3 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 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.vars=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 thrid triple 001 says that variable x_3 has exponent 1, so the first monomial consists of $x_1^3x_3$. All the remaining sets of 9 bits are interpreted the same way and we get the previous polynomial.

Notice that all binary descriptions give rise to valid polynomials. However, 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.

0.2 Regularization

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 becomes,

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

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 five runs for each λ (cf. Experimental Results includes information about these datasets).

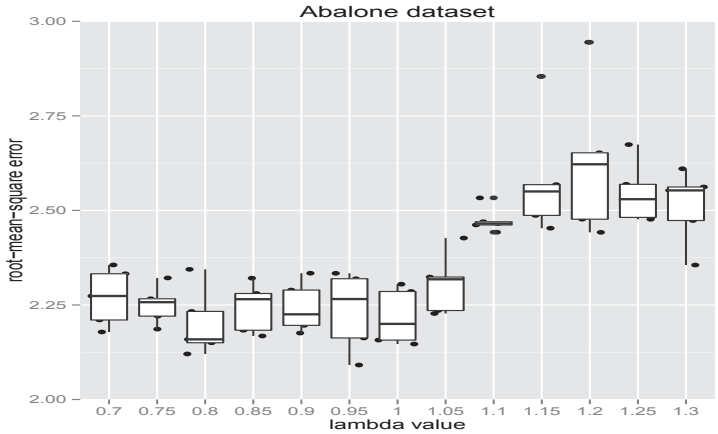

Figure 1: fitness progress for Abalone dataset

Figure 2 presents regularization results for the Auto-MPG dataset:

The typical inflection point is around $\lambda = 0.8$. We'll use the regularisation parameter with this value.

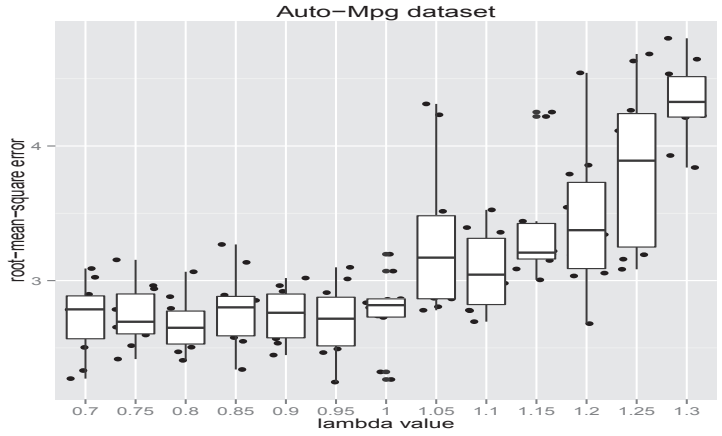

Figure 2: fitness progress for Abalone dataset

1 Experimental Results

These results were found using R's programming language [4].

To compare this paper's algorithm we applied the exact same train and test samples with other learning algorithms, namely: Linear Regression, Support Vector Machines [3], Regression Trees [7], Conditional Inference Trees [1] [6] [5] and Random Forests [2]

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 the test set in order to compute the estimated rmse. To achieve more robust results, for most datasets there were 25 runs, 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.

This dataset was used in order to verify if GApoly was able to find the polynomial relation, which the algorithm did (cf. figure 3). In this case, all runs used a population of 100 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

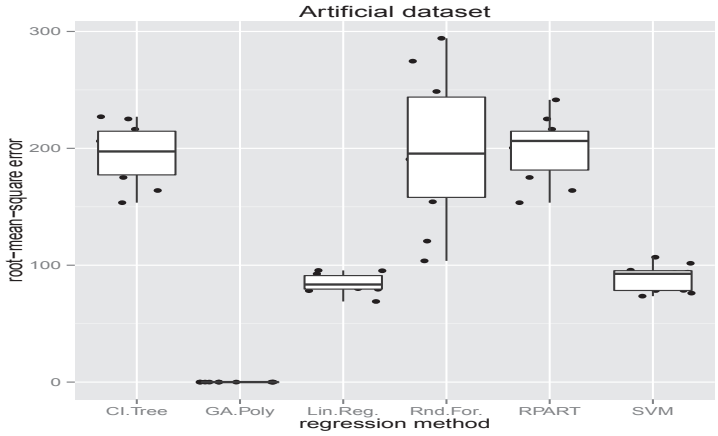

Figure 3: results for Artificial dataset

in areas of Boston. There are 13 continuous attributes and the dependent variable is the median value of owner-occupied homes in \$1000's[1]. There are 506 observations. Figure 4 presents the results for this dataset.

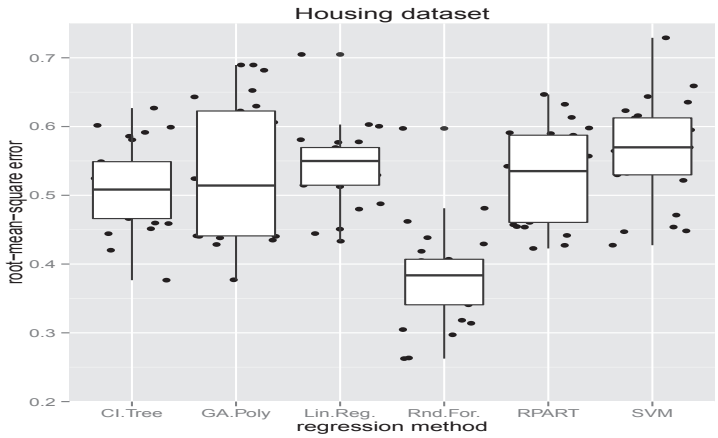

Figure 4: results for Boston Housing dataset

Just as an example of the model the GApoly algorithm outputs: in this dataset the best polynomial, with a rmse of 0.38 for the test set, was the following:

[1]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 set originally come from UCI ML repository, http://archive.ics.uci.edu/ml/

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

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.

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 5 presents the results for this dataset.

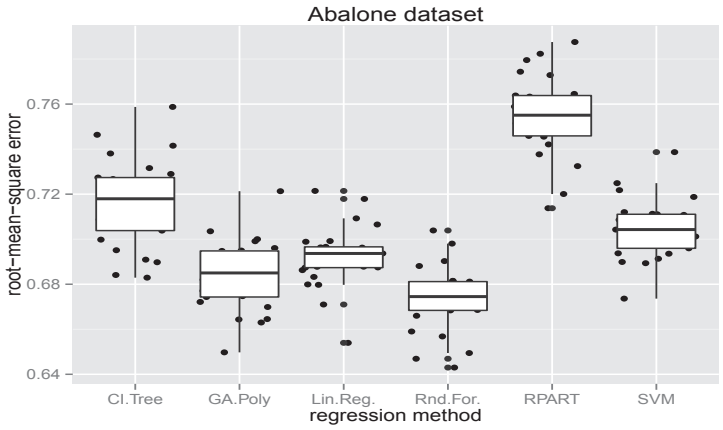

Figure 5: results for Abalone dataset

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 6 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 7 presents the results for this dataset.

1.1 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

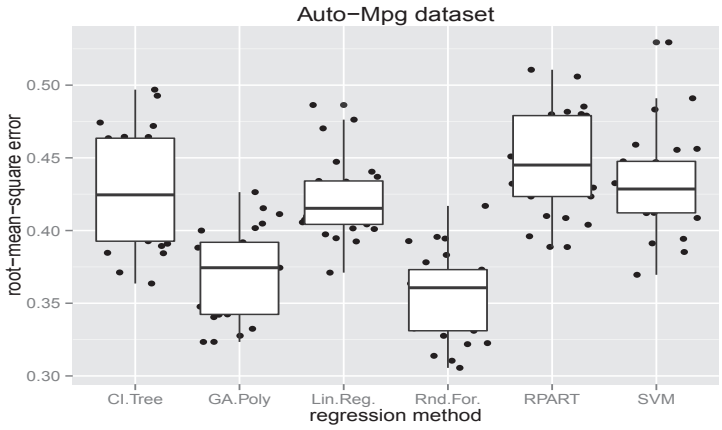

Figure 6: results for Auto-MPG dataset

current R implementation[2], placing a limit between 50 to 100 generations already achieves good results, relative to higher iteration values. Figure 8 shows a typical error evolution for the dataset Abalone given two different values for λ.

1.2 Discussion

The proposed method has competitive results comparing with some well-known regression methods. Only Random Forest outperforms GApoly systematically (it outperforms all the others) except in the Auto-MPG test and, of course, in the artificial dataset that uses a straightforward polynomial relation.

2 Bibliography

References

[1] 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.

[2]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

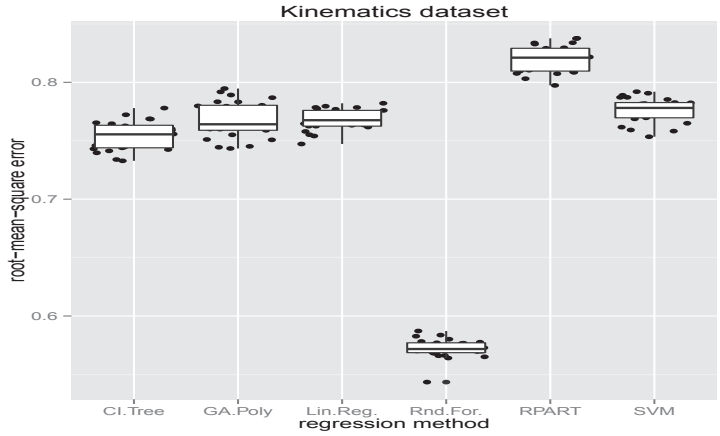

Figure 7: results for Kinematics dataset

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

[3] 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.

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

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

[6] 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.

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

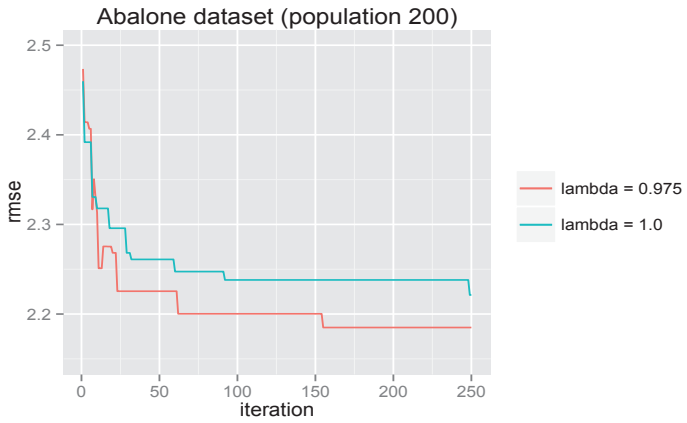

Figure 8: fitness progress for Abalone dataset