PROPER VERSUS AD-HOC MDL PRINCIPLE FOR POLYNOMIAL REGRESSION

Aleksandar Pečkov, Ljupčo Todorovski, Sašo Džeroski
Department of Knowledge Technologies
Jožef Stefan Institute
Jamova 39, 1000 Ljubljana, Slovenija
Tel: +386 1 4773144 fax +386 1 4773315
e-mail: aleksandar.peckov@ijs.si

Abstract

The paper deals with the task of polynomial regression, i.e., inducing polynomial that can be used to predict a chosen dependent variable based on the values of independent ones. As in other induction tasks, there is a trade-off between the complexity of the induced polynomial and its predictive error. One of the approaches for searching an optimal trade-off is the Minimal Description Length principle (MDL). In our previous papers on polynomial regression, we proposed an ad-hoc MDL principle. The focus in this paper is on developing a proper encoding schema for polynomials that leads to a proper MDL principle for polynomial regression. We implemented the developed MDL principle as a search heuristic in CIPER, an algorithm for inducing polynomials from data. We present an empirical comparison between the heuristics based on the ad-hoc and the proper MDL principle. The results show that proper MDL principle leads to simpler polynomials with comparable predictive error. Finally, we also propose a lower bound for the proper MDL principle that allows branch-and-bound pruning of the CIPER search space and evaluate the benefits of pruning.

1 Introduction

Regression models are able to predict the value of a dependent numerical variable from the values of independent (predictor) variables. Typical regression models include linear equations (the task is then refered to as linear regression) [5] and regression trees [6]. This paper deals with the task of polynomial regression, i.e., inducing polynomial equations for regression. Using polynomials for regression (just like using piecewise models, e.g., regression trees) can easily lead to overfitting. Namely, from interpolation theory it is known that a data set of n points can be perfectly interpolated with polynomial of $(n-1)$-th degree. A rather simplified consequence of this fact is that in the space of polynomials of arbitrary degree, we can always find a polynomial with error 0 on the training data. Many approaches for model selection have been proposed in the literature, which more or less successfully avoid the overfitting problem [7]. Minimal description length (MDL) principle is one of them.

Following the MDL principle, the quality of a model is estimated as the complexity of the model and the errors it makes on training data. The complexity of the model and the errors are measured in terms of the number of bits necessary for encoding them. Therefore, MDL measures always depend on the encoding schema chosen. While encoding schemes have been proposed for linear equations and regression (model) trees [8], up to our knowledge no encoding schema has been proposed for polynomials. The developed MDL measure is implemented within CIPER [1], an algorithm for inducing polynomials. Evaluation of the proper MDL on standard regression data sets show that proper MDL leads to simpler equations compared to the ad-hoc MDL used in the previous version of CIPER. Furthermore, we also experimented with using lower bound of the MDL measure for branch-and-bound pruning of the CIPER search space. The experimental evaluation of the branch and bound pruning shows that the lower error does not lead to much pruning for complex noisy domains.

The paper is organized as follows. In Section 2 we introduce CIPER and the ad-hoc MDL measure for polynomial regression. Section 3 presents the developed encoding of polynomials based on a proper MDL principle. We also introduce a lower bound for the MDL measure that allows branch-and-bound pruning of the search space. Section 4 presents the results of empirical evaluation. Section 5 concludes the paper and proposes directions for further research.

2 Polynomial Regression with CIPER

Every polynomial can be written in the form:

$P = C + \sum_{i=1}^{m} T_i$

where $T_i = C_i \cdot \prod {x_j}^{a_j}$, C_i , $i = 1..n$ and C are constants and $C_i \neq 0$. We say T_i is a *term* or *monomial* in P. Length of

P is $Len(P) = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{i_j}$;, size of P is $Size(P) = m$; and degree of P is $Deg(P) = Max_{i=1}^{m} \sum_{j=1}^{n} a_{i_j}$;

An example polynomial equation is $P = 1.2x^2y + 3.5xy^3$. This equation has size 2, degree 4 and length 7.

CIPER stands for Constrained Induction of Polynomial Equations for Regression. The algorithm heuristically searches through the space of possible equations for solutions that satisfy the given constraints. The output of Ciper consists of the polynomial equation, that satisfy the constraints and fit the data best.

In Ciper we have language and complexity constraints:

- *Language constraints* are in the form of sub/super polynomial. With them we are fine tuning the structure of the model. Formally, a polynomial P is sub polynomial from a polynomial Q if for every term X in P exists a term Y in Q such that the degree of every attribute in Y is bigger than or equal to the degree of the same attribute in X.

- *Complexity constraints* are constraining the complexity of a polynomial. With them we specify maximum length, maximum degree, and maximum number of terms of the polynomial.

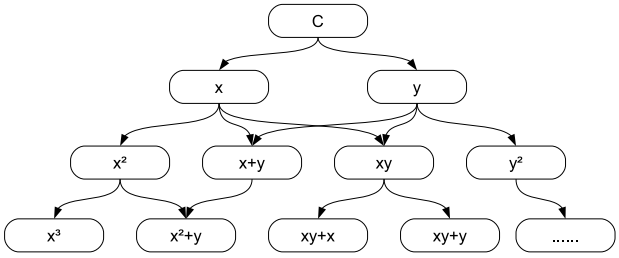
A top-level outline of the search algorithm is shown in Table 1. First, the beam is initialized either with a constant polynomial $P = C$, or with a given initial polynomial. In every search iteration, a set of polynomials is generated from the beam using the refinement operator. The coefficients before the terms are fitted using linear regression. For each of the polynomials the MDL value is calculated. The ones with the smallest MDL values are placed in the beam if the MDL value is less than the average MDL value of the polynomials in the beam multiplied with improvement coefficient. The evaluation stops when no polynomial can be generated from the beam or no polynomial is placed in the beam.

Table 1: A top-level outline of the CIPER algorithm.

procedure CIPER($Data$, $InitialPolynomial$, $Constraints$)
$InitialPol$ = FITPARAMETERS($InitialPolynomial$, $Data$)
$Q = \{InitialPolynomial\}$
repeat
  Q_r = {refinements of all equation structures in Q that satisfy the constraints $Constraints$}
  foreach equation structure $E \in Q_r$ **do**
    E = FITPARAMETERS(E, $Data$)
  endfor
  Q = {best b equations from $Q \cup Q_r$}
until Q unchanged during the last iteration
print Q

We are potentially dealing with large amounts of data that we think of as a table or as a dataset. The data can be represented in a matrix X, where the number of rows is the number of instances, and the number of columns is the number of attributes plus one (the first column is field with ones). We

Figure 1: Overview of the refinement operator



calculate the coefficients c_i of the equation as

$$c = (X^T \cdot X)^{-1} \cdot (X^T \cdot y)$$

where y is the vector of values that we are trying to predict. In this equation the multiplication is computationally expensive because of the large number of rows. If we have terms T_1,T_2, T_3 and T_4 such that $T_1 \cdot T_2 = T_3 \cdot T_4$ then the appropriate elements in their matrices are equal. We store all generated elements in the matrices for every polynomial in an array. We use it later to calculate the matrices of the subsequently generated polynomials. This optimization considerably lowers the amount of calculations.

The refinement operator increases the length of an equation by one, either by adding a first degree term or by multiplying an existing term by a variable. This operator have several good properties:

- In this way all possible equations can be generated.
- The complexity of the equation is slowly increased.
- The squared error is reduced when adding a term [10].
- Every newly generated polynomial using this operator is super polynomial of the one that it is generated from.

Because of commutativity of the addition and multiplication, one equation can be generated in more than one way (Figure 2). However, in our algorithm we avoid this problem.

We use minimum description length theory for calculating the complexities of the models. We tested two approaches for MDL, ad-hoc MDL and proper MDL. Ad-hoc MDL heuristics is

$$MDL(P) = len(P) \cdot log(m) + m \cdot log(MSE(P))$$

where P is the polynomial equation being evaluated, $len(P)$ is its length, $MSE(P)$ is its mean squared error, and m is number of training examples.

The best way of combining complexity and error is with a proper theoretical based MDL heuristics. MDL is based on two ideas:

- The model can be used to compress the data
- The better we are able to compress the data using the model, the better is the model we have induced

This gives us two ways of comparing two polynomial models. The first based on the Ad-Hoc MDL heuristics and the second based on the theoretical MDL.

3 Proper MDL for Polynomial Regression

The Minimum Description Length (MDL) Principle [12], [13], and [4], is a method for inductive inference that provides a generic solution to the model selection problem. It chooses a model that makes a trade off between well fitting the data and the complexity of the model. There are two approaches for applying MDL. The first one is called two-part code version of MDL. Here is the definition [12]:

Let $H_1, H_2, ...$ be a list of candidate models, each containing a set of point hypotheses. The best point hypothesis $H \in H_1 \cup H_2 \cup ...$ to explain the data D is the one which minimizes the sum $L(H) + L(D|H)$, where

- $L(H)$ is the length, in bits, of the description of the hypothesis;
- $L(D|H)$ is the length, in bits, of the description of the data when encoded with the hypothesis.

This approach is problematic. Its often impossible to make an ordering of the models according to the requirements of the problem. This led to the second approach called refined MDL. Here we are minimizing the sum $COMP(H') + L(D|H)$ where H' is the model (which contains more point hypotheses), $COMP$ is the complexity.

As mentioned before MDLCiper has an heuristic function based on MDL theory. We are coding the polynomial and the errors. The length of this code is the complexity of the model.

For coding the errors we use the approach of Rissanen [9] and [8] for coding a sequence of integers. The universal code for integer x is

- $UIC(0) = 1$
- $UIC(x) = 1 + log_2(2.86...) + log_2(x) + log_2(log_2(x)) + ...$

The sum stops when we encounter a negative number (the negative number is not taken).

For given precision ϵ the code of a real number x is $L(x) = UIC([\frac{x}{\epsilon}])$.

The actual coding algorithm is not given. It is only proved that the length of the function has the desired properties. The intuition behind the formula is that in order to code the integer n with a self contained prex code we need, besides $log_2(x)$ bits, also the preamble of the code, which contains the information about the length of the code ($log_2(log_2(x))$ bits), and since this preamble also needs to be coded we need $log_2(log_2(log_2(n)))$ bits for coding its length and so on. Since this code was developed on the basis of the universal prior probabilities for integers it requires $log_2(2.865064)$, which guarantees that the sum of probabilities for all positive integers equals 1. Additional 1 bit represents bounds between the encoding.

For the coding of the structure we first partition all polynomial structures in groups, such that the structures in one group have same degrees for every term. The intuition behind this is that the structure of this polynomials should have same complexity. We are actually choosing one structure from some group. If the number of all polynomials in this group is N, using the refined MDL principle, this structure has complexity $log_2(N)$. The number N can be easily calculated, for more details see [10]

For a given polynomial, we code the errors, the structure, the degrees of the polynomial, the number of terms (using UIC), and the coefficients (using fixed code), for more details see [10] The length of this code is the complexity of the model.

The new MDL heuristics have several good properties. It enabled us to do branch and bound pruning, but we will show in Section 4 that this is useful only when we have very accurate models. We are calculating complexity in a formal way, and with this avoiding the problem with ad-hoc heuristics. There is a simple way of generating more complex or more simple models, with only resizing the data [10].

For the calculation of the coefficients we do a simple linear regression. Linear regression minimizes the square error. We proved that UIC is almost convex function (there are some extreme points where UIC is not convex). This means that we are minimizing MDL, (or almost minimizing it) which allows the use of linear regression.

If we want to apply branch-and-bound pruning we need lower bound for the heuristic measures of the models. We proved that with ad-hoc MDL we don't have lower bound [10]. However with the proper MDL heuristic we can find some lower bound. Before fitting the coefficients of a generated equation, we can easily calculate the code length of the equation structure, and use this as a lower bound for the structure. Lower bound of the code for the errors is $\sum UIC(0) = N$ where N is the length of the data. Such lower bound is very small, but we try to use it.

We have implemented proper MDL based Ciper, and we present the results obtained with pruning in Section 4.

4 Empirical Evaluation

We used the same settings for the experiment as in our previous work [1].

First we present the differences between the errors that the models generated with ad-hoc and proper mdl Ciper (figure 2). We also present the differences with the model trees and regression trees. Note that Ciper generally performed better than other methods it was compared with.

We noticed that in some cases the error decreased, and in some increased. We also present the search space complexity i.e the number of generated equations. From Table 3 we notice that, generally, proper MDL generates simpler polynomials than ad-hoc MDL.

Table 2: Ad-Hoc and proper MDL, relative errors

Data set	Ad-hoc Ciper	Modal Trees	Regres. Trees	MDL Ciper	Search space
autoprice	0,4012	0,3676	0,5382	0.4434 (-)	3138
baskball	0,7959	0,7959	0,9139	0.8279 (-)	480
bodyfat	0,1679	0,1612	0,3201	0.1393 (+)	5583
cal-housing	0,5386	0,4874	0,5161	0.6004 (-)	3595
elusage	0,5216	0,5755	0,6948	0.4083 (+)	274
fried-delve	0,3196	0,2766	0,3566	0.4176 (-)	4569
house-8l	0,6159	0,5954	0,6270	0.6820 (-)	7394
housing	0,4205	0,4177	0,5050	0.4785 (-)	9853
kin-8nm	0,6025	0,6060	0,6864	0.8054 (-)	73
mbagrade	0,9167	0,9167	1,0104	1.0000 (-)	114
pw-linear	0,6273	0,3236	0,5713	0.5177 (+)	5900
quake	1,0000	1,0017	1,0051	1.0000 (=)	42
vineyard	0,7312	0,8605	0,8489	0.7197 (+)	235

The complexity of the induced polynomial is smaller in almost all cases, and in some cases a lot smaller. This generally results in increased relative error, but the general complexity of the model is always smaller.

Table 3: Ad-Hoc and proper MDL, size of the polynomial

	Ad-hoc Ciper			MDL Ciper		
Data set	Size	Len	Deg	Size	Len	Deg
autoprice	5	5	2	2	5	5
baskball	3	2	1	3	4	3
bodyfat	8	11	3	4	4	2
cal-housing	24	81	17	8	7	1
elusage	3	3	2	3	3	2
fried-delve	7	7	2	7	7	2
house-8l	35	163	13	5	13	4
housing	15	32	4	8	9	2
kin-8nm	13	16	2	3	2	1
mbagrade	3	2	1	1	0	0
pw-linear	10	12	2	6	5	1
quake	2	1	1	1	0	0
vineyard	4	4	2	2	4	4

All the experiments are done with and without branch and bound pruning. Only in one case (bodyfat) ciper searched smaller number of equations. We proved that pruning only has effect when the models make small error [10]. Even small amounts of noise can totally reduce pruning. This is however expected.

5 Conclusion

We have presented a theoretical MDL based procedure for induction of polynomial models. We have compared it with original ad-hoc based MDL and showed that the equations that are produced with the theoretical MDL are simpler than the one produced with ad-hoc MDL. We also showed that pruning of equations have effect only when the generated model is very accurate.

Future work involves modeling a MDL heuristic using Rissanen approach for calculating the complexity of a linear regression model [4]. In his work the error is not coded using universal integer codding but instead knowing that they are normally distributed he obtains the exact formula for the complexity of the model together with the data. For this to work the target variable also has to be normally distributed. But we already supposed this because we are using the method of least squares for minimization of the squared error. This makes the Rissanen approach perfect for our task.

We have successfully reduced the complexity of the modes, but on the cost of the error. Supposing a prior distribution for codding the model may ultimately lead also to more accurate models, what will be the focus of our future work.

References

[1] L. Todorovski, P. Ljubič, and S. Džeroski. Inducing polynomial equations for regression. In *Proc. Fifteenth International Conference on Machine Learning*, pages 441-452. Springer, Berlin, 2004.

[2] L. Todorovski, S. Džeroski, and P. Ljubic. Discovery of polynomial equations for regression. In *Proc. Sixth International Multi-Conference Information Society*, Volume A, pages 151-154. Jožef Stefan Institute, Ljubljana, 2003.

[3] L. Todorovski, and S. Džeroski. Theory revision in equation discovery. In *Proc. Fourth International Conference on Discovery Science*, pages 390-400. Springer, Berlin, 2001.

[4] J. Rissanen. *Lectures on statistical modeling theory*, August 2005. Available online at www.mdl-research.org.

[5] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning*, Springer, 2001. chapter 3: Linear Methods for Regression, page 41-75.

[6] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning, Springer, 2001*. chapter 9.2: Tree Based Methods, page 266-278.

[7] T. Hastie, R. Tibshirani, and J. Friedman *The elements of statistical learning*, Springer, 2001 chapter 7: Model assessment and selection, page 193-222

[8] M. Robnik - Šikonja, and I. Kononenko. Pruning Regression Trees with MDL. *European Conference on Artificial Intelligence*, page 455-459, 1998.

[9] J. Rissanen A universal prior for integers and estimation by minimum description length . *The Annals of Statistics 11(2)*, page 416-431, 1983.

[10] Ciper - online appendix. www-ai.ijs.si/∼aco/research/CIPER/CIPER.html.

[11] M. Li and P. Vitnyi. *An Introduction to Kolmogorov Complexity and Its Applications*, Springer, 1997.

[12] P. Grunwald. *A Tutorial Introduction to the Minimum Description Length Principle*.

[13] J. Rissanen. *An Introduction to the MDL Principle*; Available online at www.mdl-research.org.