Inducing Polynomial Equations for Regression

Ljupčo Todorovski, Peter Ljubič, and Sašo Džeroski

Department of Knowledge Technologies, Jožef Stefan Institute
Jamova 39, SI-1000 Ljubljana, Slovenia
Ljupco.Todorovski@ijs.si

Abstract. Regression methods aim at inducing models of numeric data. While most state-of-the-art machine learning methods for regression focus on inducing piecewise regression models (regression and model trees), we investigate the predictive performance of regression models based on polynomial equations. We present CIPER, an efficient method for inducing polynomial equations and empirically evaluate its predictive performance on standard regression tasks. The evaluation shows that polynomials compare favorably to linear and piecewise regression models, induced by standard regression methods, in terms of degree of fit and complexity. The bias-variance decomposition of predictive error shows that CIPER has lower variance than methods for inducing regression trees.

1 Introduction

Regression methods [9] aim at inducing accurate predictive models that relate the value of a target or dependent numeric variable to the values of a set of independent variables. In the last decade or so, most machine learning studies of regression as well as most state-of-the-art regression methods are concerned with inducing piecewise models. These methods partition the training set and induce a simple model in each part. Piecewise models are typically based on simple constant and linear models (as in regression and model trees [15] and MARS models [7]) or polynomials [2].

In this paper, we evaluate the usability and performance of simple models based on polynomial equations on standard regression tasks. Despite the fact that piecewise regression models prevail over simple ones in machine learning literature, we claim here that simple polynomial equations can be efficiently induced and have competitive performance with piecewise models. To approach the regression task efficiently, we develop CIPER[1], a method for inducing polynomial equations. The method performs heuristic search through the space of candidate polynomial equations. The search heuristic combines model degree of fit to the data with model complexity. We evaluate the performance of CIPER on thirteen standard regression data sets from two public repositories [1, 13]. Empirical evaluation includes comparison with standard regression methods for inducing linear

[1] The acronym CIPER stands for "Constrained Induction of Polynomial Equations for Regression".

and piecewise linear models, implemented within WEKA data mining suite [6]. We compared different methods in terms of predictive error and complexity of the induced models. We also performed empirical bias-variance decomposition of the predictive error on some of the data sets.

The paper is organized as follows. In Section 2, we present the CIPER method for heuristic search through the space of polynomial equations and discuss its relation to stepwise regression methods. Section 3 presents the results of the empirical evaluation of the proposed method and its comparison to standard regression methods. Section 4 discusses related work. Finally, Section 5 concludes the paper with a summary and directions for further work.

2 Inducing Polynomial Equations with CIPER

In this section, we present a heuristic search algorithm CIPER that searches through the space of polynomial equations and finds the one that has an optimal value of the heuristic function. First, we introduce a refinement operator that orders the space of polynomial equations. Then, we present the heuristic function used to measure the quality of each equation considered during the search along with the stopping criterion. After presenting the search algorithm based on beam search strategy, we discuss the relation of CIPER to stepwise regression methods.

2.1 The Language of Polynomial Equations

We focus here on inducing polynomial equations that can be used to predict the value of a dependent variable v_d. Given a set of variables V, and a dependent variable $v_d \in V$, a polynomial equation has the form $v_d = P$, where P is a polynomial over $V \setminus \{v_d\}$, i.e., $P = \sum_{i=1}^{r} \text{const}_i \cdot T_i$, where each T_i is a multiplicative term, r is the number of such terms, and const_i are real-valued constants. Each term is a finite product of variables from $V \setminus \{v_d\}$: $T_i = \prod_{v \in V \setminus \{v_d\}} v^{d_{v,i}}$, where $d_{v,i}$ is (a non-negative integer) degree of the variable in the term. The degree of 0 denotes that the variable does not appear in the term. The sum of degrees of all variables in a term is called the degree of the term, i.e., $\deg(T_i) = \sum_{v \in V \setminus \{v_d\}} d_{v,i}$. The degree of a polynomial is the maximum degree of a term in that polynomial, i.e., $\deg(P) = \max_{i=1}^{r} \deg(T_i)$. The length of a polynomial is the sum of the degrees of all terms in that polynomial, i.e., $\text{len}(P) = \sum_{i=1}^{r} \deg(T_i)$.

For example, consider a set of variables $V = \{x, y, z\}$, where z is chosen to be a dependent variable. The term x (that is equivalent to x^1y^0) has degree 1, the term x^2y has degree 3, while x^2y^3 is a term of degree 5. An example polynomial equation is $z = 1.2x^2y + 3.5xy^3$. It has degree 4 and length 7.

2.2 The Refinement Operator

In order to apply heuristic search methods to the task of inducing polynomial equations, we first have to order the search space of candidate equations. We introduce a refinement operator that orders this space according to equation

Table 1. The refinement operator for ordering the space of polynomial equations.

original (current) equation
$v_d = \sum_{i=1}^{r} \text{const}_i \cdot T_i$
refined equations that increase r (one for each $v \in V \setminus v_d$)
$v_d = \sum_{i=1}^{r} \text{const}_i \cdot T_i + \text{const}_{r+1} * v$, where $\forall i : v \neq T_i$
refined equations that increase d (one for each T_j and $v \in V \setminus v_d$)
$v_d = \sum_{i=1, i\neq j}^{r} \text{const}_i \cdot T_i + T_j * v$, where $\forall i \neq j : T_j * v \neq T_i$

complexity. Starting with the simplest possible equation and iteratively applying the refinement operator, all candidate polynomial equations can be generated.

Assume we measure the complexity of the polynomial equation $v_d = P$ as $\text{len}(P)$. The refinement operator increases the complexity of the equation by 1, either by adding a new linear term or by adding a variable to an existing term. First, we can add an arbitrary linear (first degree) term (that is a single variable from $V \setminus \{v_d\}$) to the current equation as presented in the first (upper) part of Table 1. Special care is taken that the newly introduced term is different from all the terms in the current equation. Second, we can increase the complexity $\text{len}(P)$ by adding a variable to one of the terms T_j in the current polynomial equation. Again, care should be taken that the changed term is different from all the other terms in the current equation. Note that the refinements of a given polynomial P are super-polynomials of P. They are minimal refinements in the sense that they increase the complexity of P by one unit.

The branching factor of the presented refinement operator depends on the number of variables $|V|$ and number of terms in the current equation r. The upper bound of the branching factor is $\mathcal{O}((|V|-1)(r+1)) = \mathcal{O}(|V|r)$, since there are at most $|V|-1$ possible refinements that increase r and at most $(|V|-1)r$ possible refinements that increase d.

The ordering of the search space of polynomial equations, defined on the set of variables $V = \{x, y, z\}$, where z is the dependent variable, is presented in Figure 1. It shows that the defined refinement operator is not optimal, in sense that each polynomial equation can be derived more than once. This is due to the commutativity of the addition and multiplication operators. An optimal refinement operator can be easily obtained by taking into account the lexical ordering of the variables in V. Then, only variables (and/or terms) with higher lexical rank should be added to the terms and/or equations. The dotted nodes in the graph in Figure 1 denote equations that would not be generated by the refinement operator that takes into account lexical order. However, the redundancy due to the sub-optimality of the refinement operator can be avoided during the search procedure, as we will point out in the following section.

While an optimal refinement operator is desired for complete/exhaustive search, it may prevent the generation of good equations in greedy heuristic search. Suppose the polynomials x and z have low heuristic value, while y has a high heuristic value and $x+y$ is actually the best. Greedy search would choose y

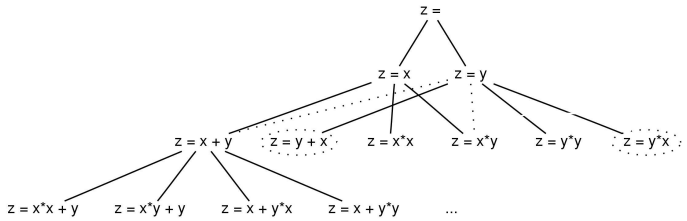

Fig. 1. The search space of polynomial equations over the set of variables $V = \{x, y, z\}$, where z is the dependent variable, as ordered by the refinement operator from Table 1. Note that for simplicity, real-valued constants are omitted from the equations.

and the optimal refinement operator that takes into account lexicographic order would not generate $x + y$.

2.3 The Search Heuristic

Each polynomial equation structure considered during the search contains a number of generic constant parameters (denoted by const_i). In order to evaluate the quality of an equation, the values of these generic constants has to be fitted against training data consisting of the observed values of the variables in V. Since the polynomial equations are linear in the constant parameters, the standard linear regression method can be used for this purpose.

The quality of the obtained equation is evaluated using a degree of fit measure that measures the discrepancy between the observed values of v_d and the values predicted using the equation. One such measure is mean squared error (MSE), calculated as: $\text{MSE}(v_d = P) = \frac{1}{m}\sum_{i=1}^{m}(v_d(i) - \hat{v_d}(i))^2$, where $v_d(i)$ is the value of v_d for the i-th training example, $\hat{v_d}(i)$ is the value of v_d for the same example, but predicted using equation $v_d = P$, and m is the number of training examples.

Ciper uses an MDL (minimal description length) based heuristic function for evaluating the quality of equations that combines the degree of fit with the complexity of the equation. In the literature, the following combination has been considered: based on Akaike and Bayesian information criteria for regression model selection [9]:

$$\text{MDL}(v_d = P) = \text{len}(P)\log m + m \log \text{MSE}(v_d = P).$$

where $\text{len}(P)$ is the length of the P, and m number of training examples. While the second term the MDL heuristic function measures the degree of fit of a given equation, the first term introduces a penalty for complexity of the equation. Through this penalty the MDL heuristic function introduces preference toward simpler equations.

2.4 The Search Algorithm

Ciper employs beam search through the space of possible equations using the search algorithm presented in Table 2. The algorithm takes as input a training

data set D containing the values of independent variables and the dependent variable v_d. The output of CIPER consists of the b best polynomial equations according to the MDL heuristic function defined in the previous section.

Table 2. A top-level outline of CIPER's beam search procedure.

```
   procedure CIPER(D, v_d, b)
 1 E_0 = simplest polynomial equation (v_d = const)
 2 E_0.MDL = FITPARAMETERS(E_0, D)
 3 Q = {E_0}
 4 repeat
 5   Q_r = {refinements of equation structures in Q}
 6   foreach equation structure E ∈ Q_r do
 7     E.MDL = FITPARAMETERS(E, D)
 8   endfor
 9   Q = {best b equations from Q ∪ Q_r}
10 until Q unchanged during the last iteration
11 print Q
```

Before the search procedure starts, the beam Q is initialized with the simplest possible polynomial equations of the form $v_d = \text{const}$. The value of the constant parameter const is fitted against the training data D using linear regression. In each search iteration, the refinements of the equations in the current beam are generated (using the refinement operator from Table 1) and collected in Q_r (line 5). In case when redundant equations are generated due to the sub-optimality of the refinement operator, the duplicate equations are filtered out from the set Q_r. Again, linear regression is used to fit the constant parameters of the refinements against the training data D (lines 6-8). Finally, at the end of each search iteration, only the best b equations, according to the MDL heuristic function, are kept in the beam (line 10). The search stops when the performed iteration does not change the beam.

2.5 Stepwise Regression

The CIPER search algorithm is similar in spirit to the forward stepwise method for linear regression [9]. As CIPER, the stepwise regression method also starts with the simplest model $v_d = \text{const}$ and sequentially adds those independent variables to the model that most significantly improve its fit to the training data. To avoid overfitting, stepwise regression methods test the significance of the MSE improvement gained by refining the current equation and do not take into account those refinements that do not lead to significant improvements. The significance of the MSE improvement is based on F statistic:

$$F = \frac{\text{MSE}(v_d = P) - \text{MSE}(v_d = P')}{MSE(v_d = P')} \cdot (m - r - 2),$$

where $v_d = P$ is the current equation, $v_d = P'$ is the candidate equation with the newly added term, r is the number of terms in the current equation, and m is

the number of training examples. The improvement is significant, if the obtained F value is greater than the 95th percentile of the $F(1, m-r-2)$ distribution [9]. Stepwise regression method proceed with greedy search by choosing the best significant improvement and stops, if no significant improvement is available.

CIPER can be viewed as a stepwise method for polynomial regression with MDL heuristic function. However, there are several other important differences between CIPER and the stepwise regression method.

The refinement operator used in CIPER is better suited for polynomial regression. While the stepwise regression method can only refine the current equation by adding a new term to it, CIPER can also add a variable to an existing term in the current equation. Using this second kind of refinement, CIPER can generate polynomials of arbitrary degree. On the other hand, to use forward stepwise method for polynomial regression, terms of degree two and more have to be precomputed and introduced as new independent variables. However, this is a serious limitation of the stepwise method, since precomputation of higher degree terms requires user to specify their maximal degree of the introduced terms and it introduces potentially huge number of independent variables. The number of independent variables is of order $\mathcal{O}(|V|^d)$, where d is the maximal degree of precomputed terms.

The huge number of precomputed higher degree terms is reflected in the high branching factor of the stepwise refinement operator. Since it adds a new term to the current equation, its branching factor equals the number of independent variables, i.e., $\mathcal{O}(|V|^d)$. Note that the branching factor of CIPER's refinement operator ($\mathcal{O}(|V|r)$) is linear with regards to the number of independent variables. The lower branching factor of the refinement operator permits the use of higher beam widths in CIPER, which is in contrast with beam width of one used for stepwise regression methods.

3 Experimental Evaluation

The main goal of the performed experiments is to evaluate the predictive performance of CIPER especially in comparison with the standard regression methods for inducing linear and piecewise models, implemented in the data mining suite WEKA [6]. The performance of the methods is evaluated on thirteen data sets from the UCI Repository [1] and another publicly available collection of regression data sets [13]. These data sets have been widely used in other comparative studies. Table 3 presents the basic properties the data sets.

3.1 Experimental Methodology and Settings

In all the experiments presented here, we estimate the regression performance on unseen examples using 10-fold cross validation. The regression performance of a model M is measured in terms of relative mean squared error (RE) defined as $\mathrm{RE}(M) = \mathrm{MSE}(M)/\mathrm{VAR}(v_d)$, where $\mathrm{VAR}(v_d)$ is the variance of the dependent

Table 3. Properties (number of variables n, number of examples m, and class variance $\mathrm{VAR}(v_d)$) of the thirteen regression data sets used in the experiments.

Data set	n	m	$\mathrm{VAR}(v_d)$
auto_price	16	159	$3.433 \cdot 10^7$
baskball	5	96	0.01173
bodyfat	15	252	69.76
cal_housing	9	20640	$1.331 \cdot 10^{10}$
elusage	3	55	566.0
fried_delve	11	40768	24.97
house_8l	9	22784	$2.792 \cdot 10^9$
housing	14	506	84.4196
kin_8nm	9	8192	0.06948
mbagrade	3	61	0.1063
pw_linear	11	200	19.92
quake	4	2178	0.03587
vineyard	4	52	18.94

variable v_d. The normalization of the MSE with the variance of v_d allows for comparison of the performance measure across different data sets.

We compare the performance of our approach based on polynomial equations to the performance of three standard regression methods implemented in WEKA [6]: linear regression, regression trees, and model trees. The tree-based models are induced with the M5' algorithm [15]. All algorithms have been used with their default parameters' settings. The default beam width in CIPER is 16. For pairwise comparison of methods, we calculate relative error reduction achieved for a given data set by using method M_1 as compared to using method M_2: $1 - \mathrm{RE}(M_1)/\mathrm{RE}(M_2)$. The statistical significance of the difference is tested using the paired t-test (with the sample size equal to the number of folds; the same folds are used for all methods) with significance level of 99%: $+/-$ to the right of a figure in the tables with results means that the difference is significant.

We also perform bias-variance decomposition of the predictive error following the Monte Carlo procedure from [8]. Following this procedure, we first randomly sample (with replacements) 100 sets from the original data set, build 100 models using method M, and use these models to obtain 100 predictions for each example in the training set. Let us denote with ${v_d}^k(i)$ the prediction of the model induced on the k-th sampled set and ${v_d}^*(i)$ the average prediction of these models. Then, the bias-variance decomposition of the squared error (SE) can be approximated as:

$$\mathrm{SE}_i(v_d = P) = ({v_d}^*(i) - v_d(i))^2 + \frac{1}{100}\sum_{k=1}^{100}({v_d}^k(i) - {v_d}^*(i))^2,$$

where the first term represent the bias and the second term represents the variance of the regression method M. We calculate then the average percentage of variance in the mean squared error.

Table 4. Predictive performance of CIPER in terms of relative mean squared error (RE), as compared to stepwise regression (with maximal polynomial degree of 1, 2, and 3) and three other regression approaches implemented in WEKA: linear regression LR, model trees MT, and regression trees RT. Sign +/− on the right hand side of the figure denotes that CIPER performed significantly better/worse.

		Stepwise regression			WEKA		
Data set	CIPER	$d=1$	$d=2$	$d=3$	LR	MT	RT
auto_price	0.1610	0.2426	0.1985	0.3966	0.2168	0.1351	0.2896 +
baskball	0.6334	0.6334	0.6397	0.6218	0.6672	0.6334	0.8351 +
bodyfat	0.0282	0.0285	0.0324	0.0286	0.0295	0.0260	0.1025 +
cal_housing	0.2901	0.3639 +	0.3339	0.3510	0.3639 +	0.2376 −	0.2664 −
elusage	0.2720	0.3604	0.2720	0.2720	0.3731	0.3312	0.4827
fried_delve	0.1021	0.2773 +	0.1128	0.0436	0.2773 +	0.0765	0.1271
house_8L	0.3793	0.6193 +	0.4370 +	16.1411	0.6191 +	0.3545	0.3932
housing	0.1768	0.2866 +	0.1676	0.1680	0.2858 +	0.1745	0.2550 +
kin_8nm	0.3631	0.5871 +	0.4390 +	0.2684 −	0.5871 +	0.3673	0.4711 +
mbagrade	0.8403	0.8403	0.8450	0.8502	0.8403	0.8403	1.0209
pw_linear	0.3936	0.2504	0.1162	0.6122	0.2377	0.1047	0.3264
quake	1.0000	0.9964	0.9964	0.9997	0.9966	1.0035	1.0102
vineyard	0.5347	0.7400	0.6674	0.7679	0.7116	0.7404	0.7207
Average	0.3980	0.4789	0.4044	1.6555	0.4774	0.3865	0.4847

3.2 Experimental Results

We present the results of the experiments in Tables 4–6. The first table compares the regression methods in terms of their predictive error, the second compares the complexity of the induced models, and the third percentage of variance in the bias-variance decomposition of the predictive error.

Predictive Error CIPER clearly outperforms linear regression and stepwise linear regression ($d = 1$) on most of the experimental data sets (and significantly on five of them). The stepwise regression methods gain higher accuracy with increasing the maximal degree of precomputed terms d, but they tend to overfit the training data — compare, for example, the results on house_8l, pw_linear, and cal_housing data sets. Although insignificant, these differences are still considerably large, especially for $d = 3$. In terms of significant differences, they are comparable to CIPER. Results of stepwise regression indicate further improvement with increasing d, however this is intractable for large data sets. Also, as we show later, stepwise regression method tend to induce more complex models. Note also that the performance of stepwise polynomial regression is very sensitive to the value of d. Selecting the optimal d value for stepwise polynomial regression is a nontrivial problem, since it can differ from one data set to another and would be, for practical reasons, guided by computational complexity issues (number of precomputed higher degree terms). Note that the computational complexity of CIPER compares favorably to the stepwise regression method with $d = 3$: the CIPER search procedure, on average, considers 10 times fewer candidate equations then stepwise regression search procedure.

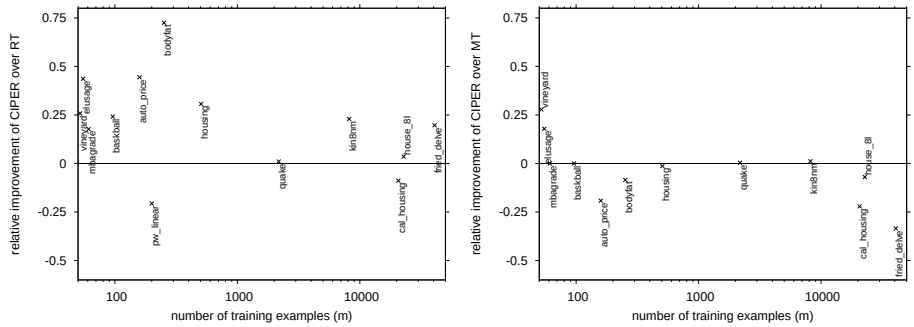

Fig. 2. The relation between number of training examples and relative accuracy improvement achieved with CIPER when compared to regression and model trees.

The overall accuracy of CIPER compares favorably to the accuracy of regression trees — CIPER is significantly better than RT in five data sets. The graph on the left-hand side of Figure 2 depicts the relative improvement of CIPER when compared to regression trees. It shows that the relative accuracy improvement is higher for smaller data sets. A possible explanation for this is that CIPER induces a single equation/model over the entire data set, as opposed to a number of partial models induced for data subsets in model trees. Finally, to our surprise, the overall accuracy of CIPER is comparable to the accuracy of model trees — note that model trees significantly outperform polynomial equations on a single data set.

Note that we also compared the predictive error of models induced using CIPER with MDL heuristic to the error of models induced using CIPER with F statistics as heuristic (used in stepwise regression methods, see Section 2.5). The results, which due to lack of space are not included in the paper, show that CIPER with MDL outperforms CIPER with F.

Model Complexity We assess the complexity of models based on polynomial equations in terms of number of constant parameters #P, length LEN (as defined in Section 2.1), and polynomial degree DEG. The complexity of tree-based models is measured in number of constant parameters in the leaf nodes #P, and number of decision nodes #DN.

Table 5 presents the results of the model complexity comparison. CIPER produces less complex polynomials than stepwise regression method with maximal degree of precomputed terms $d = 2$. Despite the (two times) higher overall degree of equations induced with CIPER, they are shorter and have two times fewer number of parameters than the equations induced with stepwise regression methods. The complexity of the polynomial models is much lower than the complexity of piecewise tree-based models. The factor of complexity reduction is of order of a hundred for both regression and model trees.

Bias-Variance Decomposition For eight of the thirteen data sets we performed bias-variance decomposition of the predictive error. As expected, the

Table 5. Complexities of the models induced with CIPER as compared to the stepwise, linear and tree-based models in terms of number of constant parameters in the equation #P, polynomial length LEN and degree DEG, as well as number of decision nodes #DN for tree-based piecewise models.

	CIPER			Stepwise ($d = 2$)			Regr. Trees		Model Trees	
Data set	#P	LEN	DEG	#P	LEN	DEG	#P	#DN	#P	#DN
auto_price	5	5	2	7	10	2	8	7	19	6
baskball	3	2	1	3	3	2	2	1	3	0
bodyfat	8	11	3	4	5	2	16	15	12	5
cal_housing	24	81	17	40	71	2	499	498	898	268
elusage	3	3	2	3	3	2	3	2	6	1
fried_delve	7	7	2	66	120	2	1919	1918	2356	538
house_8l	35	163	13	34	60	2	266	265	434	127
housing	15	32	4	29	53	2	26	25	56	18
kin_8nm	13	16	2	33	56	2	264	263	409	105
mbagrade	3	2	1	3	2	1	1	0	3	0
pw_linear	10	12	2	13	18	2	14	13	12	1
quake	2	1	1	2	1	1	1	0	10	5
vineyard	4	4	2	3	3	2	4	3	6	1
Average	10.5	26.1	4.0	18.5	31.1	1.8	232.5	231.5	324.9	82.7

Table 6. Percentage of variance in the bias-variance decomposition of predictive error of CIPER as compared to tree-based piecewise models.

Data set	CIPER	Model Trees	Regr. Trees
auto_price	2.73	12.64	10.53
bodyfat	0.02	99.43	4.98
fried_delve	<0.01	51.88	37.92
house_8l	0.01	9.80	12.56
housing	1.75	4.72	2.17
kin_8nm	0.01	38.94	26.17
pw_linear	0.32	12.21	69.04
quake	0.04	1.91	2.61

results from Table 6 show that CIPER has much lower variance than methods for inducing tree-based models.

4 Related Work

Equation discovery [10] aims at developing methods for computational discovery of quantitative laws or models, expressed in the form of equations, in collections off measured numeric data. These methods are mainly used for automated modeling of real-world systems from measurements and observations. Since they operate on numeric data, they are strongly related to regression methods. However, note that there is an important difference in focus. While equation discovery methods focus on inducing *general*, *comprehensible*, and *explanatory* models of observed systems or phenomena [11, 5], regression methods focus on the task of

inducing *accurate* model for predicting the value of a specific designated dependent variable. While the comprehensibility, generality, explanatory power of the induced model are more important for the performance of equation discovery methods than their accuracy and predictive power, the later is the most (or the only) important evaluation criterion for the regression methods.

The refinement operator in CIPER is very similar in spirit to forward stepwise methods for linear regression [9]. However, as discussed in Section 2.5, the CIPER refinement operator is much more suited for stepwise polynomial regression. Similar refinement operator has been also used in the MARS (multivariate adaptive regression splines) method [7]. The difference is however that the MARS refinement operator adds all possible *piecewise* linear terms of a form $\max(v-t,0)$ or $\max(t-v,0)$, where v is an independent variable $v \in V \setminus \{v_d\}$ and t is one of its values, to a current equation. Since each example in the training set defines a potential break point (knot) t in the piecewise linear term, the branching factor of MARS refinement operator is much higher since it also depends on the number of examples in the training set m. The branching factor of MARS refinement operator is of order $\mathcal{O}(|V|rm)$, which can be quite prohibitive for large data sets.

Finally, note that we have already presented other aspects of CIPER elsewhere. [12] focuses on the efficient implementation of the linear regression procedure (used in CIPER for constant parameter fitting) that benefits from the incremental nature of the refinement operator. In [4], we evaluate CIPER on the task of inducing ordinary differential equations.

5 Conclusions and Further Work

This paper presents CIPER, a method for efficient induction of polynomial equations that can be used as predictive regression models. CIPER employs heuristic beam search through the space of candidate polynomial equations. The search is based on a refinement operator with low branching factor that makes it much more suitable for polynomial regression compared to much complex refinement operators used in stepwise regression methods and MARS. Evaluation of CIPER on a number of standard predictive regression tasks shows that it is superior to linear regression and stepwise regression methods as well as regression trees. CIPER appears to be competitive to model trees too. The complexity of the induced polynomials, in terms of number of parameters, is much lower than the complexity of piecewise models. Finally, CIPER greatly reduce the variance of tree induction methods.

Directions for further research include integration of efficient methods for partitioning the data set in CIPER and use them to induce piecewise polynomial models (one piece for each partition). The partitioning of the data set can be based on Euclidean proximity of training examples: clustering methods can be used for this purpose as in [14, 5]. Furthermore, since linear regression method is used for fitting the model parameter in CIPER, it is fairly straightforward to develop a version of CIPER that is capable of incremental induction of regres-

sion models from numeric data streams. The development can be based on the incremental linear regression method presented in [3].

Acknowledgments This work was supported in part by the project cInQ (Consortium on discovering knowledge with Inductive Queries), funded by the European Commission under the FET arm of the IST programme.

References

1. C. L. Blake and C. J. Merz. UCI repository of machine learning databases, 1998. http://www.ics.uci.edu/∼mlearn/MLRepository.html.
2. P. Chaudhuri, M. C. Huang, W. Y. Loh, and R. Yao. Piecewise-polynomial regression trees. *Statistica Sinica*, 4:143–167, 1994.
3. Y. Chen, G. Dong, J. Han, B. Wah, and J. Wang. Multidimensional regression analysis of time-series data streams. In *Proceedings of the Twentyeighth International Conference on Very Large Data Bases*, pages 323–334. Morgan Kaufmann, 2002.
4. S. Džeroski, L. Todorovski, and P. Ljubič. Using constraints in discovering dynamics. In *Proceedings of the Sixth International Conference on Discovery Science*, pages 297–305, Berlin, 2003. Springer.
5. B. Falkenhainer and R. Michalski. Integrating quantitative and qualitative discovery in the ABACUS system. In *Machine Learning: An Artificial Intelligence Approach*, volume 3, pages 153–190. Morgan Kaufmann, San Mateo, CA, 1990.
6. E. Frank and I. H. Witten. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, San Mateo, CA, 1999.
7. J. Friedman. Multivariate adaptive regression splines (with discussion). *Annals of Statistics*, 19:1–141, 1991.
8. S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–58, 1992.
9. T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, Berlin, 2001.
10. P. Langley, H. A. Simon, G. L. Bradshaw, and J. M. Żythow. *Scientific Discovery*. MIT Press, Cambridge, MA, 1987.
11. L. Todorovski and S. Džeroski. Using domain knowledge on population dynamics modeling for equation discovery. In *Proceedings of the Twelfth European Conference on Machine Learning*, pages 478–490. Springer, 2001.
12. L. Todorovski, S. Džeroski, and P. Ljubič. Discovery of polynomial equations for regression. In *Proceedings of the Sixth International Multi-Conference Information Society (Volume A)*, pages 151–154, Ljubljana, Slovenia, 2003. Jozef Stefan Institute.
13. L. Torgo. Regression data sets, 2001. http://www.liacc.up.pt/∼ltorgo/Regression/DataSets.html.
14. L. Torgo and J. .P. da Costa. Clustered partial linear regression. In *Proceedings of the Eleventh European Conference on Machine Learning*, pages 426–436. Springer, 2000.
15. Y. Wang and I. H. Witten. Induction of model trees for predicting continuous classes. In *The Proceedings of the Poster Papers of the Eighth European Conference on Machine Learning*, pages 128–137, University of Economics, Faculty of Informatics and Statistics, Prague, 1997.