A MACHINE LEARNING APPROACH TO POLYNOMIAL REGRESSION

Aleksandar Pečkov

Doctoral Dissertation
Jožef Stefan International Postgraduate School
Ljubljana, Slovenia, October 2012

Evaluation Board:

Prof. Dr. Bogdan Filipič, Chairman, Jožef Stefan Institute, Ljubljana, Slovenia

Prof. Dr. João Gama, Member, University of Porto, Portugal

Prof. Dr. Djani Juričić, Member, Jožef Stefan Institute, Ljubljana, Slovenia

MEDNARODNA PODIPLOMSKA ŠOLA JOŽEFA STEFANA
JOŽEF STEFAN INTERNATIONAL POSTGRADUATE SCHOOL

Aleksandar Pečkov

A MACHINE LEARNING APPROACH TO POLYNOMIAL REGRESSION

Doctoral Dissertation

ALGORITMI STROJNEGA UČENJA ZA POLINOMSKO REGRESIJO

Doktorska disertacija

Supervisor: Prof. Dr. Sašo Džeroski

Co-Supervisor: Assoc. Prof. Dr. Ljupčo Todorovski

Ljubljana, Slovenia, October 2012

Contents

Abstract

In the thesis, we address the task of polynomial regression, i.e., inducing regression models based on polynomial equations, from data. We aim at improving and extending the existing approaches to learning polynomial regression models in several directions. First, we improve the existing methods for addressing the issue of over-fitting and improve the existing methods for ordering the search space of candidate polynomial equations. Second, we extend the scope of existing methods towards learning piecewise, multi-target, and classification via regression polynomial models. The central hypothesis of the thesis is that the improvements and extensions of the existing approaches are going to improve the performance of the polynomial models on regression and classification tasks. We also conjecture that their performance will be comparable to the performance of models obtained with other state-of-the-art regression and classification approaches.

To accomplish the aims and test the hypotheses, we start with performing a survey of existing research on learning regression models with focus on evaluation metrics used for regression. Then we develop new heuristics and refinement operators, and implement them into the algorithm Ciper for inducing polynomial regression models. The algorithm is capable of learning piecewise and multi-target polynomial models and polynomial models for classification via regression. Finally, we perform empirical evaluation and comparative analysis of the performance of polynomial models obtained with Ciper and the performance of models obtained with other approaches.

The results of the empirical evaluation and the comparative analysis show that the newly developed search heuristics and refinement operators lead to improved performance of the learned regression models. The performance of models induced with Ciper is comparable to the performance of models induced with other commonly used regression algorithms. Also, classification models based on multi-target polynomials have predictive performance comparable to the performance of models obtained with other classification approaches. Finally, we also show that piecewise polynomial models of limited degree perform comparable to polynomial models of higher (unlimited) degrees.

The thesis contribution to the field of machine learning is a new machine learning algorithm for inducing regression models based on polynomial equations. The algorithm is carefully designed by analyzing and comparing the performance of different methods for generating and evaluating candidate equations. The algorithm also extends the scope of polynomial regression to piecewise and multi-target regression models that can also serve well for solving classification tasks following the classification via regression approach.

Povzetek

V disertaciji obravnavamo nalogo polinomske regresije, t.j. indukcijo regresijskih modelov, ki temeljijo na polinomskih enačbah, iz podatkov. Naš cilj je namreč izboljšanje in razširitev obstoječih pristopov za učenje modelov polinomske regresije v več smereh. Najprej smo izboljšali obstoječe metode za obravnavanje problema pretiranega prilagajanja (angl. overfitting) kot tudi obstoječe metode za urejanje preiskovanega prostora polinomskih enačb. Nato smo razširili področje uporabe obstoječih metod polinomske regresije z učenjem odsekoma polinomskih modelov, večciljnih polinomskih modelov in polinomskih modelov za klasifikacijo z regresijo (angl. classification via regression). Osrednja hipoteza disertacije je, da bodo izboljšave in razširitve obstoječih pristopov izboljšale učinkovitosti oz. uspešnost polinomskih modelov na nalogah regresije in klasifikacije. Prav tako domnevamo, da bo njihova uspešnost primerljiva z uspešnostjo modelov, dobljenih z drugimi sodobnimi pristopi k regresiji in klasifikaciji.

Za doseganje ciljev in preskus hipoteze smo najprej pregledali obstoječe raziskave na področju učenja regresijskih modelov, s poudarkom na metrikah vrednotenja, ki se jih uporablja za regresijske modele. Nato smo razvili nove hevristike in izboljšane operatorje dodelave (angl. refinement operators) in jih vgradili v algoritem Ciper za učenje polinomskih regresijskih modelov. Algoritem je sposoben učenja odsekoma polinomskih in večciljnih polinomskih modelov kot tudi polinomskih modelov za razvrščanje oz. klasifikacijo z regresijo. Na koncu smo empirično ovrednotili in primerjalno analizirali uspešnost polinomskih modelov, dobljenih z algoritmom Ciper, in uspešnost modelov, dobljenih z drugimi pristopi.

Rezultati empirične ocene in primerjalne analize kažejo, da na novo razvite preiskovalne hevristike in izboljšani operatorji dodelave vodijo do izboljšanja učinkovitosti naučenih regresijskih modelov. Uspešnost modelov, naučenih z algoritmom Ciper, je primerljiva z uspešnostjo modelov, dobljenih z drugimi pogosto uporabljanimi regresijskimi algoritmi. Prav tako imajo modeli za klasifikacijo z regresijo, ki temeljijo na večciljnih polinomih, napovedno uspešnost primerljivo z uspešnostjo modelov, dobljenih z drugimi klasifikacijskimi pristopi. Na koncu smo tudi pokazali, da so odsekoma polinomski modeli z omejeno stopnjo primerljivi s polinomskimi modeli višje stopnje.

Prispevek disertacije k področju strojnega učenja je nov algoritem strojnega učenja za indukcijo regresijskih modelov, ki temeljijo na polinomskih enačbah. Algoritem je skrbno zasnovan z analizo ter primerjavo uspešnosti različnih metod za generiranje in vrednotenje kandidatskih enačb. Algoritem razširja tudi področje polinomske regresije z učenjem odsekoma polinomskih in večciljnih polinomskih regresijskih modelov, ki lahko služijo tudi za reševanje klasifikacijskih nalog v skladu s pristopom klasifikacije z regresijo.

Abbreviations

AIC	=	Akaike Information Criterion.
BIC	=	Bayesian Information Criterion.
CE	=	Classification Error.
CIPER	=	The CIPER algorithm; CIPER stands for constrained induction of polynomial equations for regression.
CV	=	Cross-Validation.
CVCIPER	=	The CIPER algorithm using the CV heuristics.
CVCIPERX	=	The CVCIPER algorithm using expansion of attributes. The algorithm induces piecewise polynomial models.
CIPERX	=	The CIPER algorithm using expansion of attributes and one of the MDL or the CV heuristic. The algorithm induces piecewise polynomial models.
J48	=	The WEKA Java implementation of the C4.5 algorithm for learning decision trees.
LR	=	The WEKA Java implementation of the Linear Regression.
MCD	=	Model Complexity by Polynomial Degree.
MCL	=	Model Complexity by Polynomial Length.
MCS	=	Model Complexity by Polynomial Size.
MDL	=	Minimum Description Length.
MDLCIPER	=	The CIPER algorithm using the MDL heuristics.
MDLCIPERX	=	The MDLCIPER algorithm using expansion of attributes. The algorithm induces piecewise polynomial models.
MSE	=	Mean Squared Error
MT	=	The WEKA Java implementation of the M5 algorithm for inducing Model Trees.
NaiveBayes	=	The WEKA Java implementation of the Naive Bayes classifier.
NML	=	Normalized Maximum Likelihood.
RMSE	=	Root Mean Squared Error.
RRMSE	=	Relative Root Mean Squared Error.
RSS	=	Residual Sum of Squares.
RT	=	The WEKA Java implementation of the M5 algorithm for inducing Regression Trees.
SMO	=	The WEKA Java implementation of support vector machines, which uses the Sequential Minimal Optimization algorithm for efficiently solving the optimization problem in training SVMs.
SVM	=	Support Vector Machines.
SMO-E1	=	SMO with polynomial kernel with exponent 1.
SMO-E2	=	SMO with polynomial kernel with exponent 2.
SMO-KR	=	SMO with a kernel with a radial basis function.
SSC	=	Search Space Complexity.
WEKA	=	Waikato Environment for Knowledge Analysis is a collection of machine learning algorithms for data mining tasks.

1 Introduction

Regression models [27] predict the value of a dependent numeric variable from the values of independent variables, also referred to as predictors (in statistics, predictors are also referred to as regressors). The regression task is the problem of inducing or learning a regression model from a table of measured values of the dependent and independent variables. The simplest approach to the regression task is linear regression, where the dependent variable is modeled as a linear combination of the predictors. More advanced regression approaches and models include regression and model trees [4] as well as multivariate adaptive regression splines (MARS) [18].

This thesis deals with the task of polynomial regression, i.e., the task of inducing a regression model in the form of a polynomial equation that predicts the value of a dependent numeric variable. We build upon an existing approach to polynomial regression, Ciper [61]. Ciper performs heuristic search through the space of candidate polynomial equations starting with the simplest polynomial and adding terms to it at each step of the search to arrive at more complex ones. Each candidate structure is matched against training data and values of constant parameters are obtained that lead to the maximal fit to the data. However, using only degree of fit to guide the search is not a good idea, since it certainly leads to over-fitting the training data [27]. Note that polynomial models can perfectly interpolate any data, since it is known that any n points can be perfectly interpolated with an $(n-1)$-th degree polynomial. To address this issue, Ciper combines the degree of fit with the polynomial's complexity to guide the heuristic search.

We address several limitations of the original version of Ciper in this thesis. First, Ciper uses an ad-hoc weighting scheme to combine degree of fit with the polynomial's complexity to obtain the value of the heuristic function for a given candidate polynomial. In contrast, we perform here an in-depth exploration of different strategies to fight the issue of over-fitting with polynomial regression models. Second, Ciper uses a straightforward simple-to-complex ordering of the search space that, combined with a specific heuristic function, might lead to under-searching the space of candidate models. In the thesis, we explore different refinement operators for ordering the space of polynomial models. Third, Ciper focuses on learning a polynomial model that predicts the value of a single dependent variable and is valid over the whole training dataset. Here, we extend the scope of polynomial regression toward multi-target regression models that can simultaneously predict several dependent variables. Also, we develop approaches to learning piecewise polynomial models. Finally, we use multi-target polynomial models on classification task by applying the classification via regression approach.

1.1 Aims and Goals

The central aim of this thesis is to overcome the three limitations of Ciper outlined above.

To address the issue of over-fitting, we implement and analyze the performance of two well-known strategies used in other machine learning approaches. The first is the minimal description length (MDL) principle [23] that introduces bias towards simpler models to avoid over-fitting. To this end, we are going to develop a proper MDL scheme for evaluating

polynomial regression models and compare its performance to the ad-hoc MDL scheme based on the Akaike information criterion [1] and Bayesian information criterion [53, 61], used in the original version of Ciper. The second strategy for avoiding over-fitting that we are going to apply in the context of polynomial regression is the use of a separate validation dataset for model evaluation. More specifically, we are going to perform a cross-validation of the candidate polynomial structure, in order to estimate its performance on unseen data.

Note that the issue of an optimal heuristic function is closely related to the ordering of the search space. In particular, combining a proper MDL (or cross-validation) scheme as a heuristic function with a simple ordering scheme of candidate polynomials might lead to under-searching the space of candidates. To address this problem, we are going to explore different refinement operators to order the space of candidate polynomial structures. As opposed to the simple refinement operator used in the original Ciper [61] that allows small elementary changes of the current polynomial structures to generate more complex ones, the new refinement operator, implemented and analyzed here, will allow for more complex changes of the polynomial structure at each step.

Moreover, we explore the issue of learning complex polynomial models. Namely, the current version of Ciper can only induce a simple regression model, i.e., a single polynomial equation that predicts the values of a single dependent variable and is valid on the whole dataset. In the thesis, we are going to develop approaches to polynomial regression capable of inducing piecewise polynomial models. We are also going to develop approaches to multi-target regression modeling, where a single model can simultaneously predict the values of multiple dependent variables.

Finally, we also aim to evaluate the developed approaches by performing an empirical comparative analysis of their performance with other state-of-the-art regression approaches. We are also going to evaluate the developed approach for multi-target regression in the context of classification by performing classification-via-regression [17]. We are going to perform the evaluation on a number of standard benchmark datasets for regression and classification.

1.2 Hypotheses and Methodology

The presented goals of the thesis can be formalized in the five central hypotheses of this thesis as follows.

H1 The newly developed heuristics for evaluating polynomial regression models, based on a proper minimal description length scheme for polynomial regression and cross-validation estimates of a model's predictive performance on unseen data, improve the performance of the learned polynomial regression models.

H2 The new refinement operator for ordering the search space of candidate polynomials that allows for larger changes of the candidate structure at each search step improves the performance of the learned polynomial regression models.

H3 The polynomial regression models, induced with the approaches developed within the thesis, have predictive performance comparable to the performance of other commonly used regression algorithms.

H4 The classification models based on multi-target polynomial regression have predictive performance equal to or better than other classification via regression approaches, as well as other classification models.

H5 Piecewise polynomial models of a limited degree achieve predictive performance comparable to the performance of polynomial regression models with unlimited degree.

We use a standard methodological framework for testing the validity of the hypotheses. First, we survey existing research on learning regression models from data. The focus of the survey is on metrics for evaluating regression models, including those that are based on the predictive error of the model, as well as metrics for measuring the complexity of the models.

Second, we develop and implement a new version of Ciper for polynomial regression that includes the newly developed heuristic measures for evaluating polynomial models, the new refinement operators, and the new approaches to learning multi-target and piecewise polynomial models.

Third, we use the new implementation of Ciper to perform an extensive comparative analysis of its performance and the performance of other state-of-the-art regression and classification approaches. The experiments include standard regression and classification benchmark problems used in related studies. The collection of datasets to be used include single-target regression, multi-target regression, and classification datasets. The analysis of the experimental results will allow us to test the validity of the five hypotheses outlined above.

1.3 Scientific Contributions

The work presented within this thesis is expected to lead to the following original contributions to the fields of machine learning and data mining:

- New heuristic functions and refinement operators for searching the space of polynomial equations and empirical comparison of their performance with the performance of the existing ones.
- Generalization of simple polynomial regression models to more complex ones: piecewise and multi-target polynomial models for regression.
- Implementation of a machine learning algorithm for inducing simple and complex regression polynomial models from data incorporating the above elements, as well as classification models via multi-target polynomial regression.

1.4 Structure of the Thesis

Chapter 2 provides a survey of the existing regression approaches and elaborative description of the previous approaches to polynomial regression. The chapter also provides an introduction to the minimal description principle and its applications to regression.

The Ciper algorithm for polynomial regression is introduced in Chapter 3. The last section of this chapter outlines the limitations of Ciper that are going to be addressed in the rest of the thesis.

Chapter 4 presents the main contributions of the thesis integrated in the new version of Ciper: the newly developed heuristic measures for evaluating polynomial models, the new refinement operators, and the new approaches to learning piecewise and multi-target polynomial models.

The next three chapters deal with the empirical evaluation and the comparative analysis of the Ciper performance. Chapter 5 introduces the experimental setup and the methodology for performing the empirical evaluation and the comparative analysis. Chapter 6 reports and discusses the results of evaluating the performance of the Ciper improvements, i.e., the newly developed heuristic measures and refinement operators, also in the context of state-of-the-art regression methods. Similarly, Chapter 7 reports and discusses the results of evaluating the performance of the Ciper extensions toward multi-target and piecewise regression models, also in the context of classification.

The final Chapter 8 provides a summary of the thesis' contributions and outlines directions for further research.

2 Background

In this chapter, we present a summary of the basic concepts used in this thesis. An overview of machine learning is given in Section 2.1. The research in this thesis builds on top of linear regression. An introduction to this subject is given in Section 2.2. More specifically this thesis is about machine learning of polynomial models. In Section 2.3, we present an introduction to polynomial models.

An introduction into model assessment and selection is presented in Section 2.4. For solving classification tasks by using the approaches presented in this thesis, we follow the classification via regression approach, presented in Section 2.5. One of the heuristic functions that we use for model selection in the thesis is based on ensemble learning. We cover this subject in Section 2.6. Another heuristic function used is based on the minimum description length principle. An introduction into this subject is presented in Section 2.7.

2.1 Machine Learning

Machine learning is a branch of artificial intelligence, concerned with the design and development of algorithms that are able to improve their behavior based on empirical data. The empirical data take a form of examples that illustrate relations between observed variables. A major focus of machine learning research is to automatically learn to recognize patterns in the examples and make intelligent decisions [22, 39].

A large part of machine learning deals with the task of modeling, i.e., building predictive models. This models predict the value of a dependent variable from the values of independent variables, also referred to as predictors. Predictive modeling problems can be divided into classification and regression problems.

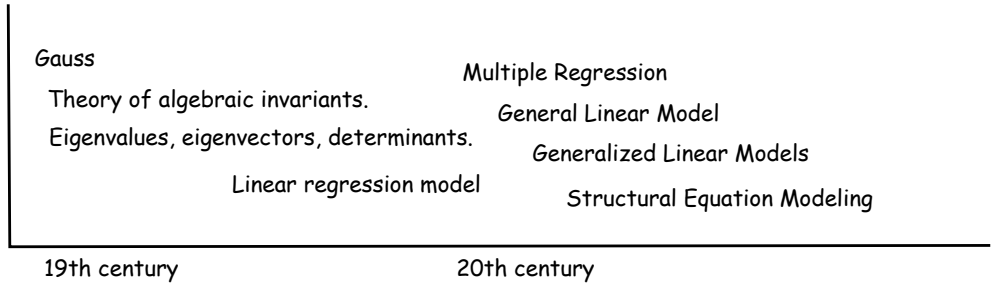
Classification problems involve predicting the values of a categorical (nominal) output variable. One or more continuous or categorical input variables can be used as predictors. There are a number of methods for solving classification problems that involve simple continuous predictors, categorical predictors, or both.

Regression problems involve predicting the value of a continuous variable from one or more continuous or categorical variables. For example, one may want to predict the selling price of a single family home from various other continuous variables and categorical (nominal) variables. *Multiple regression* can be applied for this problem, to find a linear equation that can be used to predict the selling prices from the other variables.

Within machine learning, a number of advanced statistical methods exist for handling regression and classification tasks with multiple input variables and (typically) a single output variable. These methods include Support Vector Machines (SVM) for classification and regression, Naive Bayes for classification, k-Nearest Neighbors (KNN) for classification and regression, Classification and Regression Trees (CART), Multivariate Adaptive Regression Splines (MARSplines), and others [27]. Large family of regression methods is the class of general linear regression methods, described below.

2.2 General Linear Regression

The roots of regression analysis go back to the beginnings of mathematics. The theory of algebraic invariants developed from the work of 19th century mathematicians such as Gauss, Boole, Cayley and Sylvester made the linear regression model possible. The theory identifies those quantities in systems of equations that remain unchanged under linear transformations of the variables in the system. Some of the new concepts introduced by this theory are eigenvalues, eigenvectors, determinants, and matrix decomposition methods.



The theory was soon extended to the linear regression model and correlation methods. They serve as the basis for the development of the *general linear model*. The general linear model can be seen as an extension of *linear multiple regression* for a single output variable [28].

2.2.1 Multiple Regression

The general purpose of multiple regression[1] is to quantify the relationship between several *input* variables and an *output* variable. It is assumed that the output (*dependent*) variable y is linearly related to the input (*independent*, *predictor*) variables $X_1, \cdots, X_p$ as below,

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p + \varepsilon = \beta_0 + \sum_{i=1}^{p} X_i \cdot \beta_j + \varepsilon \tag{1}$$

where ε is an unobservable random variable (the *error component*) with mean 0 and variance σ^2. The relationship described by Equation 1 is known as a *linear regression model*, where $\beta_0, \beta_1, \ldots, \beta_p$ are unknown parameters and $\sigma^2 > 0$ is an unknown *error variance*. The linearity of the model is a result of its linearity in the parameters $\beta_0, \beta_1, \ldots, \beta_p$. Transformations of the input variables (such as powers ${X_j}^d$ and products $X_j \cdot X_k$) can be included in the model without it losing its characterization as a linear regression model. The regression coefficients $(\beta_0, \beta_1, \ldots, \beta_p)$ represent the independent contributions of each input variable to the prediction of the output variable.

Typically, the parameters β are estimated from a set of training data $(x_1, y_1), \ldots, (x_N, y_N)$. Each $x_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,p})$ is a vector of feature measurements for the i-th case. The most popular estimation method is *least squares*, in which the coefficients $\beta = (\beta_0, \beta_1, \ldots, \beta_p)$ minimize the residual sum of squares

$$\mathrm{RSS}(\beta) = \sum_{i=1}^{N} (y_i - (\beta_0 + \sum_{j=1}^{p} x_{i,j} \cdot \beta_j))^2 \tag{2}$$

Denote by X the $N \times (p+1)$ matrix with each row an input vector (with a 1 in the first position, $X = (1, X_1, \ldots, X_p)$). Similarly, let $y = (y_0, y_1, \cdots, y_N)$ be the N dimensional vector

[1] In statistics, multiple regression is also referred to as linear regression

of outputs in the training set. The equation 1 can be rewriten as follows:

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} 1 & x_{1,1} & \dots & x_{1,p} \\ 1 & x_{2,1} & \dots & x_{2,p} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{N,1} & \dots & x_{N,p} \end{bmatrix} \cdot \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix} + \begin{bmatrix} \varepsilon_0 \\ \varepsilon_1 \\ \vdots \\ \varepsilon_N \end{bmatrix} \tag{3}$$

where $(\varepsilon_0, \varepsilon_1, \cdots, \varepsilon_N)$ is the vector of errors/residuals ε.

The residual sum of squares is then:

$$\text{RSS}(\beta) = (y - X\beta)^T \cdot (y - X\beta). \tag{4}$$

Assuming that X has full column rank, and hence $X^T X$ is positive definite, by setting the first derivative to zero

$$X^T \cdot (y - X\beta) = 0 \tag{5}$$

the unique solution to the minimization problem defined by Equation 2 is found to be:

$$\hat{\beta} = (X^T X)^{-1} \cdot X^T y. \tag{6}$$

The variance of residuals σ^2 is estimated using the equation:

$$\hat{\sigma}^2 = \frac{1}{(N - p - 1)} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{7}$$

where $\hat{y}_i$ is the predicted value of y at x_i.

The multiple regression model can be used to analyze only a single output variable. It can not provide a solution for the regression coefficients when the independent variables X are linearly dependent and the inverse of $X^T X$ does not exist. Different approaches presented below can be used to address these issues.

2.2.2 Multiple Output Variables

The general linear model can handle several output variables at once. The y vector of N observations of a single variable can be replaced by a Y matrix of N observations of m different Y variables. Similarly, the β vector of regression coefficients for a single Y variable can be replaced by a β matrix of regression coefficients, with one vector of β coefficients for each of the m output variables. These substitutions yield what is sometimes called the multivariate regression model, but it should be emphasized that the matrix formulations of the multiple and multivariate regression models are identical, except for the number of columns in the Y and β matrices. The method for solving for the β coefficients is also identical, that is, m different sets of regression coefficients are separately found for the m different output variables in the multivariate regression model.

The general linear model can provide a solution for the Equation 2 when the input variables are linearly dependent and thus the inverse of $X^T X$ does not exist. A non-full-rank matrix doesn't have a regular inverse. This problem is solved in the general linear model by using a generalized inverse of the $X^T X$ matrix. One way of doing this is to use regularization approaches like in *ridge regression* [29] that penalizes the magnitude of the β coefficients. The ridge regression solutions are given by the following equations:

$$\beta_\lambda = (X^T X + \lambda I)^{-1} \cdot X^T y \tag{8}$$

where λ controls the amount of penalty related to the magnitude of the coefficients.

2.2.3 Categorical Variables

The general linear model is frequently applied to analyze data that has categorical (nominal) input variables. For example, *gender* is clearly a categorical level variable. There are two basic methods by which *gender* can be coded into one or more input variables: the sigma-restricted method and the overparameterized method [28].

Using the sigma-restricted method, the males are assigned with the value -1 and the females are assigned with the value 1. The values on the resulting input variable, 1 and -1, represent a quantitative contrast between males and females. If the regression coefficient for the variable is positive, the group coded as 1 on the input variable will have a higher predicted value on the output variable, and if the regression coefficient is negative, the group coded as -1 on the input variable will have a higher predicted value on the output variable. The sigma-restricted parametrization of categorical input variables usually leads to X^TX matrices which do not require a generalized inverse for solving the minimization problem defined by Equation 2.

The overparameterized method for recoding categorical predictors is the indicator variable approach. In this method, a separate input variable is coded for each group identified by a categorical input variable. For example, females might be assigned a value of 1 and males a value of 0 on a first input variable identifying membership in the *female gender* group. Males would then be assigned a value of 1 and females a value of 0 on a second input variable identifying membership in the *male gender* group. This method of recoding categorical variables will always lead to X^TX matrices with redundant columns. Thus, it requires a generalized inverse for solving the minimization problem defined by Equation 2.

2.2.4 Generalized Linear Models

There are many relationships that cannot be described [40] by a linear equation. There are two major reasons for this.

The first reason is the *distribution of the output variable*. The output variable of interest may have a noncontinuous distribution, and thus, the predicted values should also follow the respective distribution. For example, we may be interested in predicting one of three possible discrete outcomes. The output variable can only take on 3 distinct values, and the distribution of the output variable is said to be multinomial. Or suppose we are trying to predict how many children families will have, as a function of income and various other socioeconomic indicators. The output variable *number of children* is discrete, and most likely the distribution of that variable is highly skewed (i.e., most families have 1, 2, or 3 children, fewer will have 4 or 5, very few will have 6 or 7, and so on). In this case, it is reasonable to assume that the output variable follows a Poisson distribution.

The second reason, why the linear model might be inadequate to describe a particular relationship, is that the effect of the predictors on the output variable may not be linear in nature. For example, the relationship between a person's age and various indicators of health is most likely not linear. The *generalized linear model* can be used to predict responses both for output variables with discrete distributions and for output variables which are nonlinearly related to the predictors with a *link function*.

In the generalized linear model, the relationship between the output variable y and the input variables X is assumed to be

$$y = g(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_k X_p) + \varepsilon \tag{9}$$

where g is a function. The inverse function of g, say f, is called the *link function*.

$$f(\hat{y}) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_k X_p + \varepsilon \tag{10}$$

where $\hat{y}$ stands for the expected value of y. Various link functions can be chosen, depending on the assumed distribution of the y variable:

- Identity link: $f(z) = z$
- Log link: $f(z) = log(z)$
- Power link: $f(z) = z^a$, for a given a
- Logit link: $f(z) = log(z/(1-z))$.

The parameters β are usually estimated by maximum likelihood estimation, which requires the use of iterative computational procedures.

2.2.5 Building Generalized Linear Models on Subsets of Predictors

When building generalized linear models in addition to fitting a model of the specified type using all available predictors, different methods for automatic model building can be employed that select the used predictors in different ways. For the specific type of model at hand, to build models on subsets of predictors, we can employ different methods for automatic model building. They include: forward entry, backward removal, forward stepwise, backward stepwise procedures, and best-subset search procedures. In forward methods of selection of effects (variables) to include in the model, score statistics are compared to select new significant effects.

Stepwise regression procedures involve identifying an initial model, repeatedly altering the model at the previous step by adding or removing an input variable in accordance with the stepping criteria, and terminating the search when stepping is no longer possible given the stepping criteria. For the *forward stepwise* and *forward entry methods*, the initial model always includes the regression intercept. The initial model may include one or more effects specified to be forced into the model.

In best-subset regression, the number of possible submodels increases very rapidly as the number of effects (variables) included in the model increases. The amount of computation required to perform all-possible-subsets regression increases as the number of possible submodels increases, and holding all else constant, also increases very rapidly as the number of levels for effects involving categorical predictors increases, thus resulting in more columns in the design matrix X. All possible subsets of up to a dozen or so effects could certainly theoretically be computed for a design that includes two dozen or so effects, all of which have many levels, but the computation would be very time consuming.

2.3 Polynomial Models and Polynomial Regression

A *polynomial* over variables $X_1, X_2, \ldots, X_n$ can be written in the form:

$$P = \beta_0 + \sum_{i=1}^{m} \beta_i \cdot T_i \tag{11}$$

where $T_i = \prod_{j=1}^{n} X_j^{a_{i,j}}$, $a_{i,j}$ are variable degrees, $a_{i,j} \geq 0$, and β_i, $i = 0, \cdots, n$ are constants, $\beta_i \neq 0$, $i > 0$. All T_i are referred to as *terms* or *monomials* in P. The length of P is $Len(P) = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{i,j}$, the size of P is $size(P) = m$ and the degree of P is $Deg(P) = \max_{i=1}^{m} \sum_{j=1}^{n} a_{i,j}$. An example polynomial equation is $P = 1.2X_1^2X_2 + 3.5X_1X_2^3 + 5X_1X_3 + 2$. This equation has size 3, degree 4 and length 9.

Historically, polynomial models are among the most frequently used empirical models for fitting functions. These models are popular for the following reasons:

- In mathematical analysis, the Weierstrass approximation theorem states that every continuous function defined on an interval $[a,b]$ can be uniformly approximated as

closely as desired by a polynomial function[2] [60]. They have a simple form, and well known and understood properties. They have a moderate flexibility of shapes, and they are computationally easy to use.

- Polynomial functions are a closed family. Linear transformations in the data result in a polynomial model being mapped to another polynomial model. That means that the polynomial models are not dependent on the underlying metric.

Polynomial models also have the following limitations:

- Polynomial models have poor interpolatory and extrapolatory properties. High-degree polynomials are known for oscillation at the edges of an interval[3] producing poor interpolatory properties. While polynomials may provide good fits within the range of data, the degree of fit frequently deteriorates rapidly outside the range of the data resulting in poor extrapolatory properties.
- Polynomial models have poor asymptotic properties. They have a finite response for finite values and have an infinite response if some variable takes an infinite value. Thus polynomials may not model asymptotic phenomena very well.
- Polynomial models have a shape/degree tradeoff. In order to model data with a complicated structure, the degree of the model must be high, indicating that the associated number of parameters to be estimated will also be high. This can result in highly unstable models.

Polynomial regression is a form of linear regression in which the relationship between the input variables x and the output variable y is modeled as a polynomial. Although polynomial regression fits a nonlinear model to the data, as a statistical estimation problem it is linear, in the sense that the regression function is linear in the unknown parameters that are estimated from the data. For this reason, polynomial regression is considered to be a special case of linear regression.

2.4 Model Selection and Assessment

The generalization performance of a learning method relates to its prediction capability on independent test data. Assessment of this performance is extremely important in practice. It guides the choice of learning method or model and gives us a measure of the quality of the ultimately chosen model [27].

2.4.1 Cross-Validation

Probably the most widely used method for estimating prediction error is cross-validation. Ideally, given enough data, a validation set can be set aside and used to assess the performance of our prediction model. Since data are often scarce, this is usually not possible. To solve this problem of data scarcity, K-fold cross-validation uses a part of the available data to fit the model, and a different part to test it. The data is split into K roughly equal-sized parts. For example, K-fold cross-validation when $K = 5$ is illustrated in Figure 1. For the kth part (the third in Figure 1), the model is fitted to the other $K-1$ parts of the data, and the prediction error of the fitted model is calculated for the kth part of the data. This is done for each $k = 1, 2, \cdots, K$ and the K estimates of the prediction error are combined to give the cross-validation estimate of the prediction error.

[2]The original version of this result was established by Karl Weierstrass in 1885.
[3]For feature details please see Runge's phenomenon [51].

Figure 1: A partition of a dataset used for 5-fold cross-validation. The dataset is split into 5 parts (folds). When one part of the data (in this case the third) is used for validation, the model is fit to the other four parts of the data. The prediction error is calculated on the validation part of the data. This is done in turn for each of the 5 parts and the 5 error estimates are combined.

Typical choices for K are the values of 5 or 10. The case where K equals the number of examples in the dataset is known as leave-one-out cross-validation. In this case, the cross-validation estimator is unbiased for the expected prediction error. Also, the estimate can have high variance because the training sets for each fold are very similar to one another. The computational burden is also considerable for large datasets, requiring large number of applications of the learning method.

On the other hand, for low values of K, (such as $K = 5$), the cross-validation estimate has lower variance. But the bias could be a problem, depending on how the performance of the learning method varies with the size of the training set. Overall, tenfold cross-validation is recommended as a good compromise [3].

2.4.2 Akaike Information Criterion

Akaike's information criterion (AIC) [1] is a measure of the goodness of fit of an estimated statistical model. It measures the information lost when using a specific model. As such it is based on the concept of entropy. It is usually used as a tradeoff between bias and variance in model construction.

Note that AIC is not a test of the model in the sense of hypothesis testing. It is a tool for model selection. Given a dataset, several competing models may be ranked according to their AIC, with the one having the lowest AIC being the best [7].

In the general case, the AIC is defined as

$$\mathrm{AIC} = 2p + N \cdot \ln(\frac{\mathrm{RSS}}{\mathrm{N}}) \tag{12}$$

where RSS is the estimated residual sum of squares of the fitted model, p is the number of parameters and N is the size of the sample. AIC rewards the goodness of fit and includes a penalty for the number of estimated parameters. This penalty discourages over-fitting.

It is important to realize that the AIC value assigned to a model is only meant to rank competing models and to determine which is the best among the given alternatives. The absolute values of the AIC for different models have no meaning. Only relative differences can be ascribed meaning.

When the sample size is small, a modified version of AIC, namely AICc should be used. AICc is defined as:

$$\mathrm{AICc} = \mathrm{AIC} + \frac{2p(p+1)}{N-p-1} \tag{13}$$

We can notice that AICc has a greater penalty for the extra parameters as compared to AIC. Since AICc converges to AIC as n gets large, AICc can be employed regardless of sample size [7].

2.4.3 Bayesian Information Criterion

The *Bayesian information criterion (BIC)* or *Schwarz criterion* [53] is a criterion for model selection among a class of parametric models with different numbers of parameters. The generic form of BIC is

$$\mathrm{BIC} = -2 \cdot N \cdot \ln \mathrm{RSS} + p \ln(N). \tag{14}$$

where RSS is the estimated residual (sum of squares) of the fitted model, p is the number of parameters and N is the size of the sample.

When estimating model parameters, it is possible to increase the model likelihood by adding parameters, which may result in over-fitting. The BIC resolves this problem by introducing a penalty term for the number of parameters in the model.

Given any two estimated models, the model with the lower value of BIC is the one to be preferred. BIC generally penalizes free parameters more strongly than the Akaike information criterion, though this depends on the size of N and the relative magnitude of N and p.

2.4.4 Statistical Comparisons of Predictive Models

A *t-test* is a statistical hypothesis test, most commonly applied when the test statistic follows a normal distribution. It is commonly used to test if the means of two normally distributed populations are equal. Such tests are often referred to as unpaired t-tests. The t-test can also be applied to test whether the difference between two responses measured on the same statistical unit has a mean value of zero. Such a t-test is often referred to as a paired t-test or paired difference test.

An *F-test* is any statistical hypothesis test in which the test statistic has an F-distribution under the null hypothesis. It is commonly used to test if the means of two normally distributed populations that have the same standard deviation are equal. The F-test plays an important role in the analysis of variance (ANOVA). Another usage of the F-test is to test if a proposed regression model fits the data well. Exact F-tests mainly arise when the models have been fitted to the data using the least squares approach.

The *Wilcoxon signed-rank test* is a non-parametric statistical hypothesis test for the case of two related samples or repeated measurements on a single sample. It can be used as an alternative to the paired Student's t-test (described above) when the population cannot be assumed to be normally distributed [67].

The *Friedman test* [20] [19] is a non-parametric statistical test used to detect differences in treatments across multiple test attempts. The procedure involves ranking each row (or block) together, then considering the values of ranks by columns. A classical example of usage is in the case when we have n judges (datasets) that rate k different predictors (models build by machine learning algorithms). We can determine if any predictors are ranked consistently higher or lower than the others.

2.5 Classification via Regression

It is possible to use regression methods to solve classification tasks. In order to apply the continuous prediction technique of regression models to discrete classification problems, an approximation of the conditional class probability function can be considered. During classification, the class whose model yields the greatest approximated probability value is chosen as the predicted class [17]. This approach is called *classification via regression*.

For each of the classes (class values) of the target attribute from the original dataset, a new dataset is created. Each derived dataset is the same as the original, except for the class value which is set to 1 or 0 depending on whether that instance has the appropriate original class value or not. Next, a regression algorithm is used to generate a model for each of the datasets. For a specific instance, the output of one of these models constitutes an approximation to the probability that this instance belongs to the associated class. An instance of unknown class is processed by each of the regression models. The class whose regression model gives the highest value is chosen as the predicted class.

Frank et al. exploit this approach by using model trees as regression functions. Model trees are binary decision trees with linear regression functions at the leaf nodes: Thus,

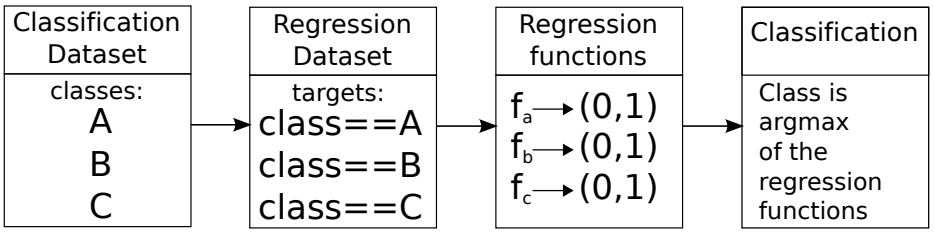


Figure 2: The process of classification via regression.

they can represent any piecewise linear approximation to an unknown function [44]. The construction and use of model trees is clearly described in Quinlan account of the M5 algorithm. An implementation of M5, called M5' is described by Wang and Witten.

Frank et al. show that the classifiers based on the smoothed model trees generated by M5' are significantly more accurate than the pruned decision trees generated by C5.0[4]. This is true for the majority of datasets, particularly those with numeric attributes. Frank et al. also achieve surprisingly good results using basic linear regression for the regression functions.

2.6 Ensemble Learning

Supervised learning algorithms are commonly described as performing the task of searching through a hypothesis space to find a suitable hypothesis that will make good predictions for a particular problem. Even if the hypothesis space contains hypotheses that are very well-suited for a particular problem, it may be very difficult to find a good one. Ensembles combine multiple hypotheses to form a better hypothesis.

Evaluating the prediction of an ensemble typically requires more computation than evaluating the prediction of a single model, so ensembles may be thought of as a way to compensate for the poor performance of individual learning algorithms by performing a lot of extra computation. Fast algorithms, such as decision tree learners are commonly used with ensembles, although slower algorithms can benefit from ensemble techniques as well.

An ensemble learning method is itself a supervised learning algorithm, because it can be trained and then used to make predictions. The trained ensemble represents a single hypothesis. This hypothesis need not necessarily be contained within the hypothesis space of the models from which it is built. Thus, ensembles have more flexibility in the functions they can represent. While this flexibility can enable them to over-fit the training data more than a single model would, in practice ensemble techniques tend to reduce problems related to over-fitting of the training data.

Ensembles tend to yield better results when there is a significant diversity among the models [38] [58]. Many ensemble methods seek to promote diversity among the models they combine [6]. Although perhaps non-intuitive, randomized algorithms can be used to produce stronger ensembles than deterministic algorithms. Using a variety of learning algorithms has been shown to be more effective than using techniques that attempt to over-simplify the models in order to promote diversity [21].

2.7 The Minimum Description Length Principle

The Minimum Description Length (MDL) Principle is a relatively recent method for inductive inference that provides a generic solution to the model selection problem. MDL was introduced by Jorma Rissanen in 1978. It is based on the following insight: any regularity in the data can be used to compress the data, describing it using fewer symbols than the

[4]C5.0 is the successor of C4.5 [44].

number of symbols needed to describe the data literally [23]. The more regularities there are, the more the data can be compressed. In this way, learning is equal to finding regularity: The more we are able to compress the data, the more he have learned about it.

Despite its simplicity, the idea represents a drastically different view of modeling. The model class has to be such that its members can be described or encoded in terms of a finite number of symbols, e.g., binary. The length of the compressed data is a measure of its complexity.

The Kolmogorov complexity K of a string is the length of the string's shortest description in some fixed description language. This is the length of the shortest program that produces the string. The most problematic issue about Kolmogorov complexity is that K is not a computable function, i.e., there is no program which takes a string s as input and produces the integer $K(s)$ as output. But for a specific domain, there sometimes is a way of calculating complexity. This is an objective measure for the complexity of a model and yielding a formal way of comparing different models. This is the beginning of the Minimum Description Length (MDL) theory.

A fundamental construct in the MDL approach to modeling is the *universal model* defined by the considered class of models. The code length of the data obtainable by a special universal model is called the *stochastic complexity* of the data, relative to the model class.

2.7.1 Stochastic Complexity

Let y^n be a sequence of n observations. A model class is defined as a set of probability densities $\{f(y^n; \theta) : \theta\}$ over such sequences of observations, parametrized by a finite-dimensional parameter vector θ. The maximum likelihood estimator of the parameter vector is denoted by $\hat{\theta}(y^n)$ gives us (for a sequence of observations y^n) the value of the probability vector θ that maximizes the probability of observing y^n.

The normalized maximum likelihood (NML) density for a model class parametrized by a parameter vector θ is defined by

$$\hat{f}(y^n) = \frac{f(y^n; \hat{\theta}(y^n))}{C^n} \tag{15}$$

where C^n is a normalizing constant and $\hat{\theta}(y^n)$ is the maximum likelihood estimator.

The NML code is said to be universal in that it gives the shortest description of the data achievable with a given model class[5]. It thus defines the *stochastic complexity* of the data for the model class. The MDL principle chooses the model class for which stochastic complexity is minimized.

NML can be seen as seeking a balance between fit and complexity. The numerator measures how well the best model in the model class can represent the data. The denominator penalizes complex model classes. The logarithm of the denominator, $ln(C^n)$, is named *parametric complexity* of the model class[6]. One of the most active areas of research within the MDL framework is the problem of unbounded parametric complexity. Which makes it impossible to define the NML density for models from the geometric, Poisson, and Gaussian families [23].

For model classes with unbounded parametric complexity, Rissanen [47] proposes to use a two-part scheme. The range of the data is first encoded using a code based on a universal code for integers. Then the data is encoded using NML, taking advantage of the restricted range. Stine [59] gives an analysis of similar schemes, where the range of the parameters is restricted instead of the data. The weakness in such solutions is that they typically result in two-part codes that are not complete, i.e., the corresponding density integrates to less than one.

[5]For further details please see [54] and [49]

[6]For more details please see [23] and [49]

2.7.2 Stochastic Complexity of a Linear Regression Model

Rissanen [48] describes an elegant renormalization scheme where the hyperparameters defining the range of the data are first optimized. A second normalization is then performed such that the resulting code is complete. This *renormalized* NML can be used for model selection in linear regression and denoising.

Based on this, Rissanen provides a formula for calculating the stochastic complexity of a linear regression model generated by using the method of least squares

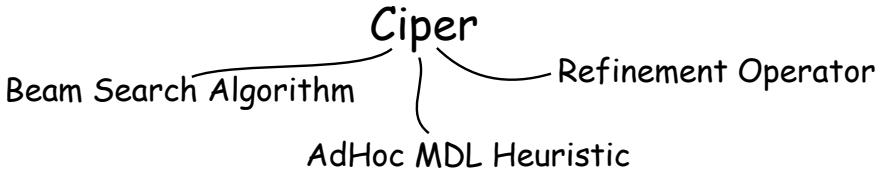
$$W = \min_{\gamma}\{(N-p)log(\hat{\tau}) + p\ log(N\ \hat{R}) + (N-p-1)log(\frac{N}{N-p}) - (p+1)log(p)\} \qquad (16)$$

where the γ index goes through all the possible subsets of variables involved in the linear regression, p is the number of elements in γ, N is the size of the dataset, $\hat{\tau}$ is the maximum likelihood estimation of the model error, and $\hat{R} = \frac{1}{N}\hat{\beta}^T(X^TX)\hat{\beta}$ (where $\hat{\beta} = (X^TX)^{-1}X^Ty$ and $X = (1, X_1, \ldots, X_p)$ is the data matrix[7]). The stochastic complexity of the model is then $2W$ (For further details, please see [26], [48], and [50]).

Intuitively, this corresponds to the length of the code necessary to encode the errors of the linear regression model together with the constant parameters of the linear model. The two are closely related and thus the constant parameters are not encoded separately or with the model structure, which is what is usually done in machine learning algorithms when using the MDL principle in an ad-hoc manner.

[7] For details on multiple regression, please see Section 2.2.1.

3 CIPER: Constrained Induction of Polynomial Equations for Regression



In this chapter, we present CIPER [61], a machine learning algorithm for finding polynomial equations. CIPER is a heuristic algorithm that searches through the space of polynomial equations and finds one (or several equations) that satisfy a given set of constraints and have an optimal value of the given heuristic function. It uses beam search to heuristically search through the space of possible equations for ones that fit the data best. CIPER uses ideas from stepwise regression, best-subset regression and machine learning.

As mentioned in Section 2.3, a *polynomial* over variables $X_1, X_2, \ldots, X_n$ can be written in the form:

$$P = \beta_0 + \sum_{i=1}^{m} \beta_i \cdot T_i \tag{17}$$

where $T_i = \prod_{j=1}^{n} X_j^{a_{i,j}}$, β_i ,$i = 1..m$ and β_0 are constants, and $\beta_i \neq 0$. We also defined the length of P as $Len(P) = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{i,j}$, the size of P as $size(P) = m$, the degree of a term T_i as $Deg(T_i) = \sum_{j=1}^{n} a_{i,j}$, and the degree of P as $Deg(P) = \max_{i=1}^{m} Deg(T_i)$.

Historically, polynomial models are among the most frequently used empirical models for fitting functions. They are popular because they have a simple form; they have well-known and understood properties; they have a moderate flexibility of shapes; and they are computationally easy to use. Also, every continuous function defined on an interval $[a,b]$ can be uniformly approximated as closely as desired by a polynomial function (the Weierstrass approximation theorem)[60] made them even more desirable.

The consequence of the Weierstrass theorem for polynomial regression is that there are many polynomials that can provide good fit to a given finite dataset. One way to cope with this problem is to constrain the space of candidate polynomial equations. CIPER makes use of two classes of constraints for this purpose.

- *Language constraints* are given in the form of a sub/super polynomial of the polynomial we are looking for. They restrict the structure of the possible polynomial models. Formally, a polynomial P is a sub-polynomial of a polynomial Q if for every term X in P, there exists a term Y in Q, such that the degree of every variable in Y is larger or equal than the degree of the same variable in X. For example, xy^2 is a sub-polynomial of x^2y^4z.

- *Complexity constraints* limit the complexity of a polynomial. With them we specify the maximum length, maximum degree, and maximum number of terms in the polynomial. For example, one might be interested in equations of degree at most 3 with at most 4 terms.

3.1 Polynomial Regression as Search

A refinement operator implements a function that takes as input an equation structure and generates a new equation structure by modifying the old one. The original CIPER refinement operator increases the length of an equation by one, either by adding a first degree term or by multiplying an existing term with a variable (Figure 3). Starting with the simplest equation (a constant), and iteratively applying this refinement operator, all polynomial equations can be generated. In the case of sub/super polynomial constraints, we start with an initial polynomial equation: By applying the refinement operator all super/sub polynomials of the initial equation can be generated.

Given an expression $x+y$, we can refine it in two ways. First, we can include a new linear term yielding $x+y+z$. Second, we can replace an existing term in the expression (e.g, x) by multiplying it with a variable (e.g, z), yielding a new expression (e.g, $xz+y$).

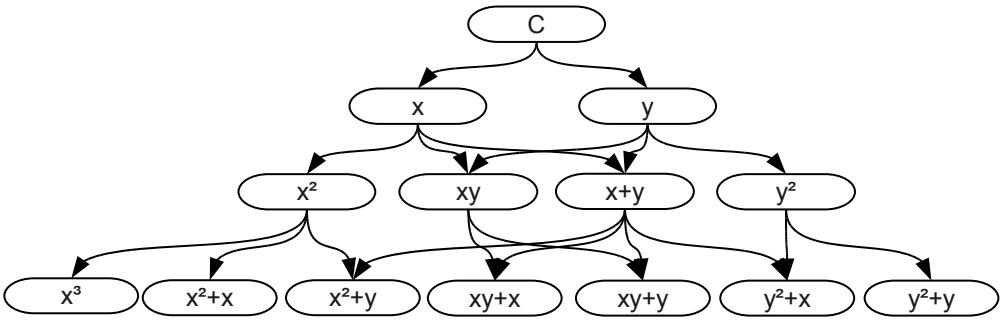


Figure 3: A lattice of polynomial equation structures generated by the original CIPER refinement operator. Equation length is increased by one in each refinement step.

Assume we measure the complexity of the polynomial equation as its length. The refinement operator increases the complexity of the equation by one, either by adding a new linear term or by adding a variable to an existing term. First, an arbitrary linear (first degree) term can be added to the current equation. 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 by adding a variable to one of the terms. Again, care should be taken that the resulting term is different from all the other terms in the current equation. Note that the refinements of a given polynomial are super-polynomials of it. They are minimal refinements in the sense that they increase its complexity by one unit.

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

3.2 The Ad-Hoc MDL Heuristic

To evaluate equations, we calculate different measures of the degree of fit of an equation to a given dataset. Two measures commonly used for regression are the mean squared error (MSE) and the multiple correlation coefficient R. Various other types of prediction error measures are often used. These include mean absolute error, maximum absolute error, and root mean square error ($\mathrm{RMSE}=\sqrt{\mathrm{MSE}}$). Most of these are well-known from statistics. In the machine learning literature, the measure RE, defined as $\mathrm{RE}^2=\frac{\mathrm{MSE}}{\sigma^2}$, where σ^2 is the variance of the dependent variable, is often used to evaluate the performance of regression approaches. The normalization with the variance allows for comparisons of performance across different datasets.

The original CIPER implementation uses the AdHoc heuristic, defined as follows

$$\text{AdHoc}(P) = \text{len}(P) \cdot \log(m) + m \cdot \log(\text{MSE}(P)) \tag{18}$$

where P is the polynomial equation being evaluated, $\text{len}(P)$ is its length, $\text{MSE}(P)$ is its mean squared error, and m is the number of training examples. This evaluation function is based on the Akaike and Bayesian information criteria[1] for regression model selection [61]. The second term of the AdHoc heuristic function measures the degree of fit of a given equation to the data and the first term introduces a penalty for the complexity of the equation. With this penalty, the AdHoc heuristic function introduces a preference toward simpler equations.

3.3 The Algorithm

CIPER searches through the space of possible equations by using a beam search algorithm. At any point in time, it maintains a set of b best possible equations (the beam) that satisfy the imposed constraints. The output of CIPER consists of the final contents of its beam, i.e., the best polynomial equations, accordingly to the ad-hoc heuristic function defined in the previous section.

The top-level outline of the CIPER algorithm is shown in Table 1. First, the beam is initialized either with the simplest polynomial equation $P = C$ (where C is constant), or with a user specified minimal polynomial. In every search iteration, a set of new, more complex polynomials is generated from the polynomials in the beam by using a refinement operator.

The coefficients before the terms in a polynomial are fitted by using the method of least squares. For each of the generated polynomials, the value of the AdHoc heuristic is calculated. At the end of the iteration, the equations with smallest heuristic values are retained in the beam.

The search evaluation stops when the refinement operator can not generate new equations. It can also stop if the content of the beam is unchanged in the last iteration. Such a situation occurs when every polynomial generated in the last iteration has a worse heuristic score estimate than the polynomials already in the beam.

Table 1: A top-level outline of the CIPER algorithm. Q is the set of b best equations (the beam) and Q_r is the set of refined equations.

```
procedure CIPER (Data, InitialPolynomial, Constraints)
InitialPolynomial = FITPARAMETERS(InitialPolynomial, Data)
Q = {InitialPolynomial}
repeat
  Q_r = refinements of equation structures in Q that satisfy
    the given Constraints
  foreach equation structure E ∈ Q_r do
    E = FITPARAMETERS(E, Data)
  endfor
  Q = {best b equations from Q ∪ Q_r}
until Q unchanged during the last iteration
print Q
```

Some optimizations for fitting the coefficients of the generated polynomial structure can be introduced. The data are represented as a matrix X, where the number of rows is the number of instances, and the number of columns is the number of terms (r) plus one (the first

[1] For a short introduction to the Akaike and Bayesian information criteria, see Sections 2.4.2 and 2.4.3.

column is filled with ones). The least squares estimate for the coefficients $\beta = (\beta_0, \beta_1, \cdots, \beta_r)$ of the equation is

$$\beta = (X^T \cdot X)^{-1} \cdot (X^T \cdot y) \tag{19}$$

where y is the vector of values we are trying to predict[2].

In Equation 19, the multiplication is computationally expensive because of the large number of rows. Let T_1 and T_2 be terms in equation A, T_3 and T_4 terms in equation B, such that $T_1 \cdot T_2 = T_3 \cdot T_4$. Then the appropriate elements in the matrices $X_A^T \cdot X_A$ and $X_B^T \cdot X_B$ are equal. We store all generated elements from the matrices $X^T \cdot X$. We reuse them for calculating the matrices of the subsequently generated polynomials. This optimization considerably lowers the computational cost of CIPER at the expense of some memory.

3.4 Evaluating CIPER

The empirical evaluation by Todorovski et al. shows that CIPER outperforms linear regression and stepwise linear regression on most of the experimental datasets considered. The stepwise regression methods gain accuracy with increasing the maximal degree of precomputed terms d, but they induce much more complex models and tend to over-fit the training data. The results of stepwise regression indicate that further improvement is possible if we increase the degree further: however, this is intractable for large datasets.

Stepwise regression tends to produce more and more complex models as d increases, and the performance of stepwise polynomial regression is very sensitive to the value of d. Selecting the optimal d value is a nontrivial problem, since it can differ from one dataset to another. For practical reasons, the selection would be guided by computational complexity issues (the number of precomputed higher degree terms) [61].

The overall accuracy of CIPER is comparable to the accuracy of regression trees. The relative accuracy improvement is higher for smaller datasets. The latter provide insufficient statistical support for a number of partial models derived from parts of the dataset (as in regression trees and model trees), but sufficient support for a single equation over the entire dataset (as in CIPER) [61].

3.5 The Limitations of CIPER

The limitations of CIPER include a limited refinement operator, an ad-hoc heuristic function, and the inherent limitation of the language of polynomial equations. In this thesis, we set out to develop improvements and extensions of CIPER that overcome these limitations. Before describing the improvements, we discuss briefly each of the above mentioned CIPER limitations.

CIPER has a very limited refinement operator. Adding a term to a linear (in the parameters) equation always decreases its error (at least on training data). However, replacing a term with a more complex version doesn't necessarily decrease the error of the equation. If we add z to $x+y$, yielding $x+y+z$, we will reduce the error of the equation. However, if we replace x with xz, yielding $xz+y$, xz need not be strongly correlated with x and the replacement might actually increase the error of the equation. This motivated us to develop a improved refinement operator.

The CIPER heuristic function is based on the AIC and BIC criteria and implements the MDL principle in an ad-hoc fashion. Using a theoretically sound MDL heuristic would be more appropriate. We focused our research on finding an MDL heuristic for polynomial equations. We also experimented with using a cross-validation error estimate as a heuristic.

In cross-validation, we construct a model by omitting each fold and using it as a validation set. The resulting models learned with each of ten folds left out, are combined in

[2]For a short introduction to multiple regression, see Section 2.2.1.

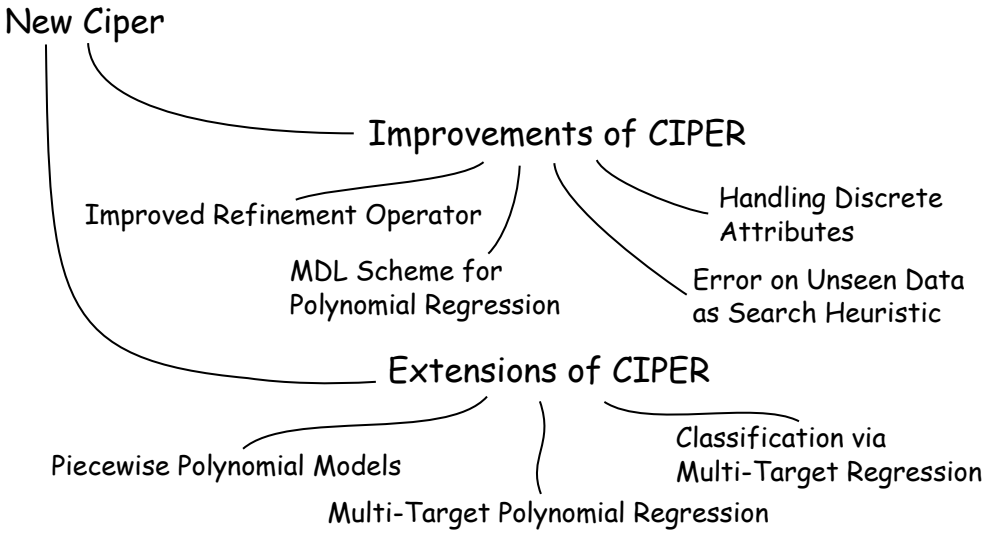
an ensemble yielding one final polynomial model. The sum of polynomial models is again a polynomial model, which doesn't necessary hold for other model structures(such as regression trees).

Polynomial equations have certain inherent limitations, as mentioned in Section 2.3. To overcome them, we will extend the CIPER algorithm to learn piecewise models by introducing new attributes (based on the existing ones) that split the instance space into several parts.

We generalize the CIPER algorithm to support multiple targets. We use a similar approach as for general linear models[3]. This multi-target version of CIPER is then used for classification via regression.

[3]For a short introduction to the general linear model, see Section 2.2.2.

4 The New Ciper Algorithm



4.1 Improvements of Ciper

In this section, we introduce several improvements of the original Ciper algorithm. We first introduce the improved refinement operator, a major improvement over the old one. We next introduce the improved MDL heuristic, followed by a heuristic based on a cross-validation estimate of prediction error. Finally, we describe the handling of categorical attributes.

4.1.1 Improving the Refinement Operator

Adding a term to a linear (in the parameters) equation always decreases its error (at least on training data). However, replacing a term with a more complex version of it (multiplied by a variable) doesn't necessarily decrease the error of the equation. If we add y to x, yielding $x+y$, we will reduce the error of the equation. However, if we replace x with xy, the replacement might actually increase the error of the equation.

This has motivated us to modify the refinement operator in Ciper. Besides the two types of refinements considered in the original version of Ciper, we introduce a third one. We take a term in the equation, make a copy, multiply the copy with a new variable and add the product back to the equation. For example, with the new operator $x+y$ can be refined to $x+y+xy$ by copying the term x, multiplying it with y, and adding the newly obtained term xy to the equation.

The old refinement operator always increases the complexity of an equation by one. As illustrated in Figure 4, the new refinement operator can increase the complexity of an equation considerably. Because of this, we introduce an extra simplification step in Ciper.

For every equation in the beam, we try removing each of its terms: if this yields an equation with a better heuristic value, as compared to the original equation, we add the newly formed simplified equation to the beam.

The extra simplification step is the last type of refinements that we use within the new refinement operator. Note that the new refinement operator has now four types of refinements: the original two, i.e., adding a single variable term (e.g., $x \to x+y$) and multiplying a term (e.g., $x \to x \cdot y$), the third refinement type that adds a multiplied term (e.g., $x \to x+x \cdot y$) and the last type of simplification refinement (e.g., $x+y+x \cdot y \to x+x \cdot y$).

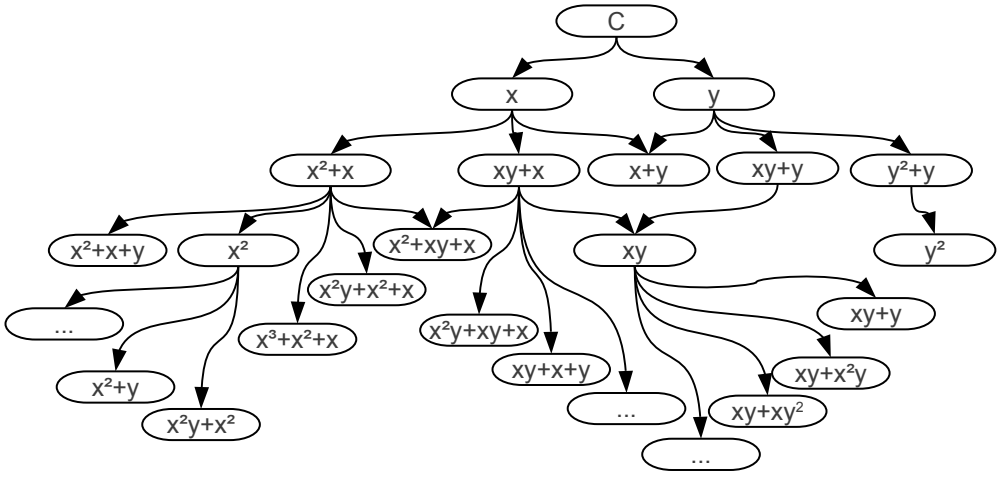


Figure 4: The improved Ciper refinement operator. The length of an equation can increase by more than one in each refinement step.

The branching factor of the new refinement operator depends on the number of variables V and the number of terms in the current equation r. The upper bound of the branching factor is $O(V+V \cdot r+V \cdot r) = O(V \cdot r)$, since there are at most $V+V \cdot r$ possible refinements that increase the number of terms r and at most $V \cdot r$ possible refinements that increase only the degree d, but not the number of terms.

The partial complexity of the extra simplification step is simply the number of terms of the equation $O(r)$. The simplification step is repeated until there are better equations generated with it. In theory, if this step is repeated r times, we can produce a constant equation. Hence the complexity of the extra simplification step is $O(r^2)$.

In Section 3.1, we have calculated the branching factor of the old refinement operator as $O(V \cdot r)$. As we can see, the old and the new refinement operators have similar branching factors i.e., $O(V \cdot r)$ and $O(V \cdot r)+O(r^2)$, respectively.

4.1.2 MDL Scheme for Polynomial Regression

4.1.2.1 Encoding the Polynomial Structure

In order to encode the structure of a polynomial, we follow the refined MDL[1] approach [23]. We first partition the space of candidate models into subgroups $\mathscr{H}_c$ of models with equal complexity c. A particular model $H \in \mathscr{H}_c$ can be then encoded using $N = \log|\mathscr{H}_c|$ ($\log$ stands for the binary logarithm) bits, where $|\mathscr{H}_c|$ denotes the number of models in the class $\mathscr{H}_c$.

In the case of polynomials, we partition the space of candidate polynomial structures into classes at several levels. At the highest level, we group together the candidate polynomials with the same length l and the same size m. Recall that for a polynomial $p(x_1,x_2,\cdots,x_n) = \beta_0+\sum_{i=1}^{m}\beta_i\prod_{j=1}^{n}x^{a_{i,j}}$, the size m is defined as the number of terms m, and the length l is

[1]For a short introduction to the MDL theory, please see Section 2.7.

defined as $Len(P) = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{i,j}$ (note also that $m \leq l$). We refer to these classes as $G(m,l)$. For example $G(1,1)$, contains polynomial structures with a single term, which has to be linear (length 1). Similarly, $G(1,2)$ contains structures with a single term of second degree, while $G(2,4)$ contains structures with two terms, and the degrees of the terms can be up to four (since the length is 4).

At the second level, we partition each $G(m,l)$ in subclasses with fixed term degrees $G'(a_1,a_2,\cdots,a_m)$. All polynomials in this subclass have m terms with degrees $a_1 \geq a_2 \geq \cdots \geq a_m$. Note that $\sum_{i=1}^{m} a_i = l$. For example, $G(2,4)$ can be broken into two subclasses of $G'(1,3)$, and $G'(2,2)$. The first subclass $G'(1,3)$ contains polynomial structures with one linear term and one third-degree term, while the second subclass $G'(2,2)$ contains polynomial structures with two second-degree terms.

Now we have to calculate how many sub-classes G' there are in a single $G(m,l)$ class and also calculate how many polynomial structures there are in each $G'(a_1,a_2,\cdots,a_m)$ class.

The number of structures in each G', $|G'(a_1,a_2,\cdots,a_m)|$ can be easily calculated using a procedure roughly depicted in Figure 5. Given the degree of the first term a_1, we have to choose a_1 variables from the set $\{x_1,x_2,\cdots,x_n\}$, where variables can appear in the selection more than once. Thus, the number of possibilities for the first term equals the number of combinations with repetition, where we select a_1 elements from a set of n elements. This number equals $\binom{n+a_1-1}{a_1}$. Continuing the same reasoning for all m terms, we find the number of possible structures in $G'(a_1,a_2,\cdots,a_m)$ to be $\prod_{i=1}^{m} \binom{n+a_i-1}{a_i}$. However, if there are several a_i values that are equal, we will encounter the same term many times, which means that the above formula over-estimates the number of possible structures. The remedy is to divide the number with the factorial of repetitions observed in the tuple. For example, when dealing with the case $G'(5,5,3,2,2,2)$, we have to divide the above product with 2!3!, since a fifth degree term appears twice (2!) and a second degree term appears three times (3!). Note also that each multiplicative term decreases by 1 for each degree value repetition (see Figure 5).

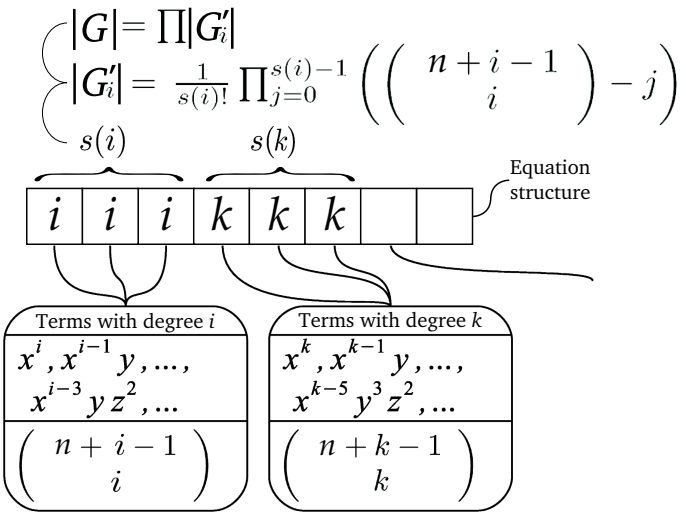


Figure 5: Calculating the number of polynomial structures in $G'(a_1,a_2,\cdots,a_m)$. At the bottom, we have the sets of terms (two sets are depicted, one with terms of degree i and one with terms of degree k). In the middle layer, they are combined into equation structures, where $s(i)$ and $s(k)$ denote the numbers of repetitions of the i and k values respectively.

Having the number of equation structures in each G' class, we now turn to the problem of calculating the number of G' classes within each $G(m,l)$. The size of G grows according to the recursive formula $|G(m,l)| = |G(m-1,l-1)| + |G(m,l-m)|$. The first additive term corresponds to the cases when the G' classes contain linear terms (there is an a_i with value 1), while the second corresponds to the cases when all terms in the G' classes have a degree at least 2 (all $a_i > 1$). In the first case, when removing the linear term, we obtain polynomials

with $m-1$ terms and length $l-1$. In the second case, we can remove one variable from each of the terms, which leads to polynomials with the same number of terms (m) and length $m-l$. Take for example $G(2,4)$, mentioned above: $|G(2,4)|=|G(1,3)|+|G(2,2)|$, $G(1,3)$ contains structures with one term that has degree 3, up to degree two, and $G(2,2)$ contains structures with two terms, up to degree 2. Figure 6 depicts the relationship between the G and G' classes of polynomial structures.

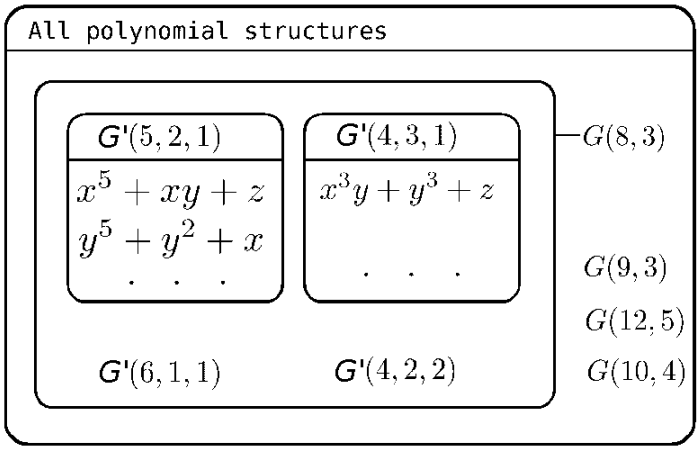


Figure 6: A general overview of the partitioning of polynomial structures. The small sets correspond to G' classes (e.g., the set $G'(5,2,1)$). In turn, we group them into larger classes of structures G that have the same length and size.

The recursive formula always leads to one of the two simple G classes, which contain a single G' subclass. The first is $G(1,l)$ with a single subclass $G'(l)$, where $|G(1,l)|=|G'(l)|$. The second simple class is $G(l,l)$ with a single subclass $G'(1,1,\cdots,1)$, where 1 is repeated l times. In this case, $|G(l,l)|=|G'(1,1,\cdots,1)|$. Finally, note that the recursive formula above can lead to the illegal situation $G(m,l-m)$ with $m>l-m$; in such cases $|G(m,l-m)|=0$.

Now, having this partitioning and the number of polynomials in each partition, we can decompose the code for each candidate polynomial into four components. First, we have to encode its length l and for this we need $\log l+2\log(\log l)$ bits (the second double logarithm term is necessary, since we do not know the magnitude of l in advance). Second, we encode the number of terms m, for which we need $\log l$ bits (remember that $m \leq l$). Third, we can identify a particular G' class within the class $G(m,l)$ using $\log|G(m,l)|$ bits. Finally, we identify the specific polynomial structure within G' using $\log|G'(a_1,a_2,\cdots,a_m)|$ bits. Putting these four components together gives us the final formula:

$$L(H) = 2\log l + 2\log(\log l) + \log|G(n,l)| + \log|G'(a_1,a_2,\cdots,a_n)| \tag{20}$$

for the number of bits necessary to encode the polynomial structure.

4.1.2.2 Extending MDL to Support Binary Attributes

Note that for any binary attribute[2] X, $X^d = X$, for an arbitrary degree d. Taking this into account, the number of structures in each G' subclass has to be adjusted. Note also that we need an additional bit to encode the fact that a polynomial structure contains a binary attribute.

From here on, we assume that a polynomial structure contains one or more binary attributes. Consequently, there are polynomials in the $G'(a_1,a_2,\cdots,a_m)$ class that are equivalent because of the binary attributes. Thus this number should be recalculated. First, we

[2]Binary attributes are part of piecewise polynomial models, see Section 4.2.1 and of handling discrete attributes, see Section 4.1.4

need one bit of information to identify if the polynomial has binary attributes. If it does not, then the number is calculated just like before.

Let the number of binary attributes be n_b. If the polynomial has binary attributes, then for every monomial, we need to encode the number of binary attributes that that monomial contains. For this we need $\sum \log a_i$ bits of information (as the number of binary attributes is smaller then the degree of the monomial). Let the i-th monomial contain b_i, binary attributes. The number of monomials that have degree $a_i - b_i$ is $\binom{n-n_b+a_i-b_i-1}{a_i-b_i}$. The number of monomials that have degree b_i for which every variable is contained only once, is $\binom{n_b}{b_i}$. The number of monomials that have degree a_i and have b_i binary attributes is $\binom{n-n_b+a_i-b_i-1}{a_i-b_i} \cdot \binom{n_b}{b_i}$. The number of possible structures in $G'(a_1, a_2, \cdots a_m)$ is

$$|G'(a_1, a_2, \cdots, a_m)| = \prod_{i=1}^{m} \binom{n - n_b + a_i - b_i - 1}{a_i - b_i} \cdot \binom{n_b}{b_i}.$$

If there are several (a_i, b_i) values that are equal, we will encounter the same term many times. The remedy is to divide the number with the factorial of repetitions observed in the tuple.

4.1.2.3 The Complete Scheme

The complexity of the polynomial structure, explained above, plus the stochastic complexity of the linear regression model[3] gives the total complexity of the model:

$$\mathrm{MDL}(H) = \mathrm{L}(H) + 2\,\mathrm{W}(H, D) \tag{21}$$

The stochastic complexity of the linear regression model is calculated with equation 16 proposed by Rissanen and explained in Section 2.7.

An illustration of the stochastic complexity of the linear model and the complexity of the polynomial structure is given in Table 2. For this examples we use the *auto-price* dataset [16]. This data-set has a single target and 33 attributes, which include *curbWeight*, *width*, *length* and the other attributes that appear in Table 2.

In the case of a multi-target model[4], the structure of the polynomial does not change, the complexity of encoding it is the same for the multi-target case and the single target case. We need, however, to sum the stochastic complexities of the linear models for each of the targets with the complexity of the structure they yield the total complexity of the multi-target model.

An illustration of the stochastic complexity of the linear models and the complexity of the polynomial structure for the multi-target case is given in Table 3. For this example we use the *sigmeareal* dataset [11]. This dataset has 2 targets and 8 attributes, which include X, Y, *angle*, *visualAngle*, and *minDistance*.

4.1.3 Using Error on Unseen Data as Search Heuristic

As an alternative to the MDL scheme described above, we propose to use as a heuristic the error of a polynomial equation as estimated on unseen data. The procedure is very similar to the procedure for cross-validation described in Section 2.4.1. We split the train set into 10 parts. We build a model on 9 parts and we use the 10th part for validation. Let the squared relative error on the 9 parts for a given model be $reTrain^2$ and the squared relative error on the 10th part be $reTest^2$. For each model, we keep the tuple $(reTrain^2, reTest^2)$. The heuristic value of the model is $\max(reTrain^2, reTest^2)$ (the smaller the better). We modify the beam search procedure in CIPER so that a (child) model generated from this model (using the refinement operator) can enter the beam only if its heuristic value is smaller than $\min(reTrain^2, reTest^2)$.

[3] A short introduction to stochastic complexity is given in Section 2.7.

[4] The multi-target approach is introduced in Section 4.2.2.

Table 2: Stochastic complexity of the linear model ($2W$) and the complexity of the polynomial structure L and the total complexity of the polynomial model (MDL) for several polynomial models learned on the *auto-price* dataset. The components needed to calculate L according to Equation 20, i.e., l, $|G|$, and $|G'|$, as well as those needed to calculate W according to Equation 16, i.e., $\hat{\tau}$ and $\hat{R}$, are also given. The number of independent variables is $n = 15$ and the number of examples is $N = 159$ for this dataset.

Polynomial	l	$\|G\|$	$\|G'\|$	$\hat{R}$	$\hat{\tau}$	W	L	MDL
$p_1 = -15378.219 +$ $+ 10.90 \cdot curbWeight$	1	1	15	$1.58 \cdot 10^8$	2629.59	1846.87	5.91	3699.65
$p_2 = -61636 +$ $+ 8.04 \cdot curbWeight +$ $+ 812.25 \cdot width$	2	1	105	$1.59 \cdot 10^8$	2511.70	3616.94	11.21	3721.22
$p_3 = -35220 +$ $+ 0.13 \cdot width \cdot curbWeight +$ $+ 398 \cdot width$	3	1	1800	$1.59 \cdot 10^8$	2451.6	1849.57	16.81	3715.95
$p_4 = -6030.87 +$ $+ 0.011 \cdot wheelBase^2 \cdot length +$ $- 0.001 \cdot length^3 +$ $- 0.416 \cdot height^2 +$ $+ 52.19 \cdot engineSize +$ $- 1097 \cdot stroke +$ $+ 0.021 \cdot curbWeight \cdot horsepower$	12	11	$1.73 \cdot 10^{11}$	$1.59 \cdot 10^8$	2509.54	1926.2	51.96	3904.38

Table 3: The stochastic complexity of the linear model for the first target - $2W_1$, for the second target - $2W_2$, the complexity of the polynomial structure L and the total complexity of the multi-target polynomial model MDL for several models learned on the *sigmareal* dataset.

Polynomial	W_1	W_2	L	MDL
$p_1 = (-2.469, -7.052) + (6.115, 17.507) \cdot visualAngle$	1043.96	2112.63	4.58	6317.77
$p_2 = (-8.068, -22.136) + (9.709, 27.190) \cdot visualAngle +$ $+ (0.141, 0.381) \cdot minDistance$	920.20	1961.86	8.41	5772.51
$p_3 = (-0.224, -0.612) + (4.139, 5.215) \cdot visualAngle^2 +$ $+ (1.531, 4.010) \cdot visualAngle^3 +$ $+ (0.316, 1.262) \cdot angle \cdot visualAngle^2 +$ $+ (-0.111, -0.320) \cdot Y \cdot visualAngle^2 +$ $+ (0.0001, 0.0003) \cdot Y^2 +$ $+ (0.0006, 0.1670) \cdot X \cdot visualAngle^2$	723.31	1637.07	43.57	4764.33
$p_4 = (0.067, 0.020) + (0.000, 0.000) \cdot X^3 +$ $+ (0.000, 0.000) \cdot Y^3 +$ $+ (0.000, 0.000) \cdot X^2 \cdot Y +$ $+ (0.000, 0.000) \cdot X \cdot Y^2 +$ $+ (0.002, 0.006) \cdot X \cdot Y$	1373.95	2526.66	39.15	7840.36

While we could consider only the error on the validation set, the CIPER search procedure generates many new models from a single model. The probability that some of them will have a smaller error on the validation set than the parent model, just by coincidence, is high. To avoid over-fitting we also consider the error on the train set. The error on the train set should be smaller than the error on the validation set. If this is not the case, we will not generate refined models from this model: The refinement operator will not consider

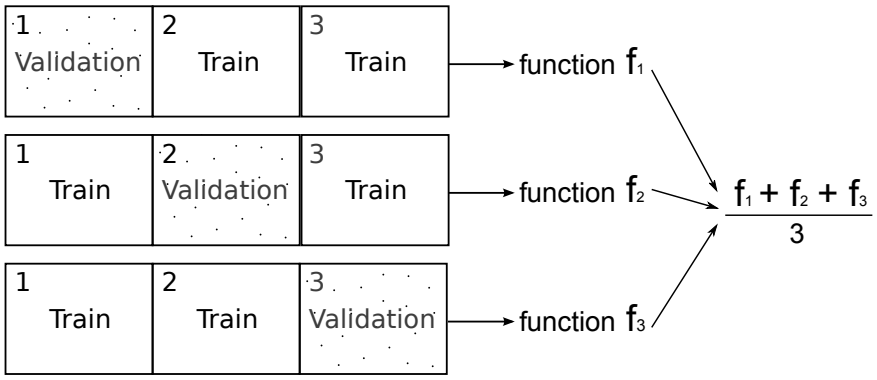


Figure 7: Constructing an ensemble model by CVCIPER. After leaving each fold out, a model is constructed from the remaining folds by using the CV heuristic to select models.

this polynomial in the future generations. We can still over-fit, but the probability of this would be smaller.

After we build 10 models, leaving each of the ten parts out, we need to make the final model. We average all the models to make one final model. The final model can be viewed as ensemble of polynomial models[5]. The process of constructing the ensemble is illustrated in Figure 7.

In the case of multi-target models[6] the heuristic is the tuple

$$(\sum_{i=1}^{t} \frac{reTrain^2}{t}, \sum_{i=1}^{t} \frac{reTest^2}{t})$$

where t is the number of targets. Because we use relative error, the worse value of the values in the heuristic is at 1 and bigger. The best value of the heuristic is at 0. We will refer to this heuristic as CV heuristic.

The top-level outline of the CVCIPER algorithm is shown in Table 4. First the *Data* is split into several folds. For each F of the folds, first the beam Q is initialized either with the simplest polynomial equation $P = C$, or with a user specified polynomial. In every search iteration, a set of new, more complex polynomials is generated from the polynomials in the beam by using a refinement operator.

The coefficients before the terms in a polynomial are fitted on the train data *F.train* of the fold F. For each of the generated polynomials T, we calculate the value of the heuristic ($T.ErrorOnTrain, T.ErrorOnValidation$). At the end of the iteration, the equations with the smallest heuristic values are retained in the beam.

The search evaluation for this fold F stops when the refinement operator can not generate any new equations. It can also stop if the content of the beam is unchanged in the last iteration. Such a situation occurs when every polynomial generated in the last iteration has a worse CV heuristic estimate than the polynomials already in the beam.

The best equation e from the beam Q is appended to the set of best equations E. One such equation is generated for each of the folds. The average of this equations is the final polynomial equation R.

4.1.4 Handling Discrete Attributes

The original CIPER can only handle numeric attributes. To allow for discrete (nominal) attributes, we use a procedure that first converts the discrete attributes to binary attributes. For this, we use the overparametrized method, described in Section 2.2.3.

[5]For short introduction of ensemble learning, please see Section 2.6.

[6]The multi-target approach is introduced in Section 4.2.2.

Table 4: A top-level outline of the CVCIPER algorithm. Q is the set of best b equations (the beam) and Q_r is the set of refined equations. S is the set of ten folds that the *Data* is split into. Each fold F is a tuple (*Train*, *Validation*) where *F.Train* is used for both training and validation, while *F.Validation* is used only to validate the current equation. For each of these tuples, a polynomial equation is generated. E is the set of these equations. The resulting polynomial equation R is the average of all equations in E.

```
procedure CVCIPER (Data, InitialPolynomial, Constraints)
S = SPLITINTOFOLDS(Data, numberOfFolds)
E = ∅
foreach F ∈ S do
 InitialPolynomial = FITPARAMETERS(InitialPolynomial, F.Train)
 Q = {InitialPolynomial}
 repeat
   Q_r = refinements of equation structures in Q that satisfy
     the given Constraints
   foreach equation structure T ∈ Q_r do
     T = FITPARAMETERS(T, F.Train)
     T.ErrorOnTrain = EVALUATEERROR(t, F.Train)
     T.ErrorOnValidation = EVALUATEERROR(t, F.Validation)
   endfor
   Q = {best b equations from Q ∪ Q_r}
 until Q unchanged during the last iteration
 e = best equation in Q
 E = E ∪ {e}
endfor
R = AVERAGE(E)
```

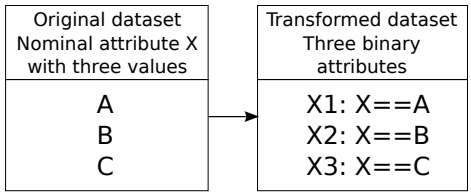


Figure 8: Handling discrete attributes.

Let the values of a discrete attribute X be $v_1, v_2, ..., v_k$, where k stands for the number of different values the attribute X has. We replace the discrete attribute X with k binary attributes in the transformed dataset. Each binary attribute X_i is defined as $X_i \equiv (X == v_i)$, meaning:

$$\begin{cases} X_i = 1, & \text{if } X == v_i \\ X_i = 0, & \text{otherwise} \end{cases}$$

The original CIPER handles binary attributes as numeric. Note however, that each binary attribute X has the property that $X^d = X$ for an arbitrary degree d. Thus, in the new CIPER we modified the search procedure to consider only the first degree of the binary variables, since all the higher degrees are equivalent to the first degree.

4.2 Extensions of CIPER

In this section, we describe the extensions of CIPER in directions of learning piecewise polynomial models, multi-target polynomial regression, and classification via regression.

4.2.1 Piecewise Polynomial Models

In order to extend CIPER models in the direction of model trees, we consider piecewise polynomial models. Unlike trees, we partition the space along each continuous dimension before the search begins and not during the search. The partitioning is not dependent on the target.

For a given array A of real values, we partition the array in k arrays $A_1, A_2, \cdots, A_k$. Let m_i be the mean of the values in the array A_i. Then the heuristic H for this partitioning is given with:

$$H(A_1, A_2, \cdots, A_k) = k \cdot \left(\sum_{j=2}^{k} \left(\frac{m_j - m_{j-1}}{2}\right)^2 + \sum_{x \in A} \min_{j=1}^{k} (x - m_j)^2\right) \tag{22}$$

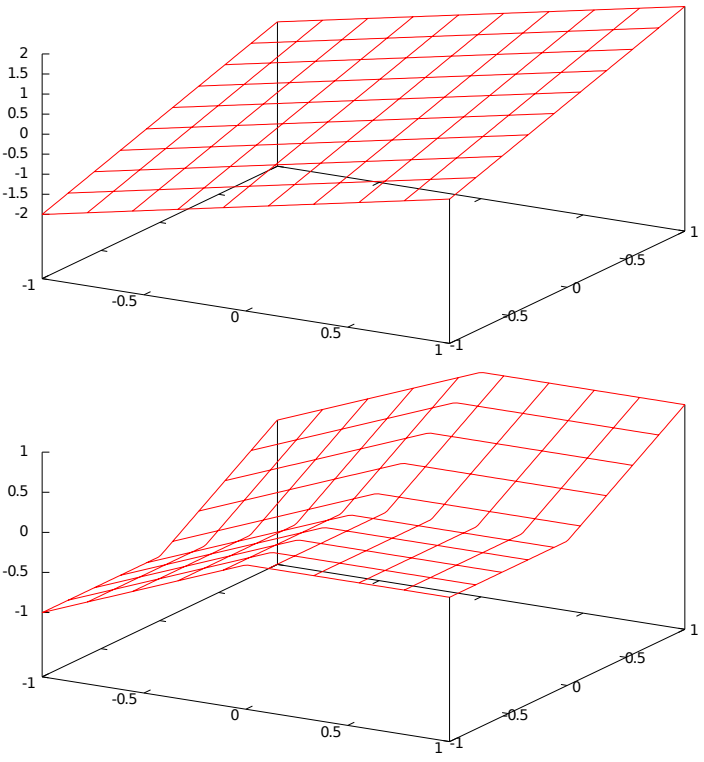


Figure 9: Example of a piecewise model. The upper image is a graph of a normal (non-piecewise) linear (polynomial) function $f(x,y) = x + y$. The second image is a graph of a piecewise linear (polynomial) function $f(x,y) = \frac{1-\text{sgn}(x)}{2} \cdot x + \frac{\text{sgn}(y)+1}{2} \cdot y = \frac{x+y}{2} + \frac{y\text{sgn}(y) - x\text{sgn}(x)}{2}$.

If the above heuristic value of the k - partitioning $A_1, A_2, \cdots, A_k$ is considerably smaller than $\sum_{x \in A}(x - mean(A))^2$, then we say that this array can be split into k sets: $A_1, A_2, \cdots, A_k$. If such a partitioning exists, then we say that the number k is a *possible split* for the array A. Given k, the partitioning $A_1^s, \cdots, A_k^s$ is called *best* if k is a *possible split* and

$$H(A_1^s, A_2^s, \cdots, A_k^s) \leq H(A_1, A_2, \cdots, A_k) \tag{23}$$

for any other partitioning $A_1, A_2, \cdots, A_k$ of the array A.

Let $\text{P}_k(A) = (A_1^s, A_2^s, \cdots, A_k^s)$, i.e., $\text{P}_k(A)$ be a function that finds the best partitioning of an array A into k arrays.

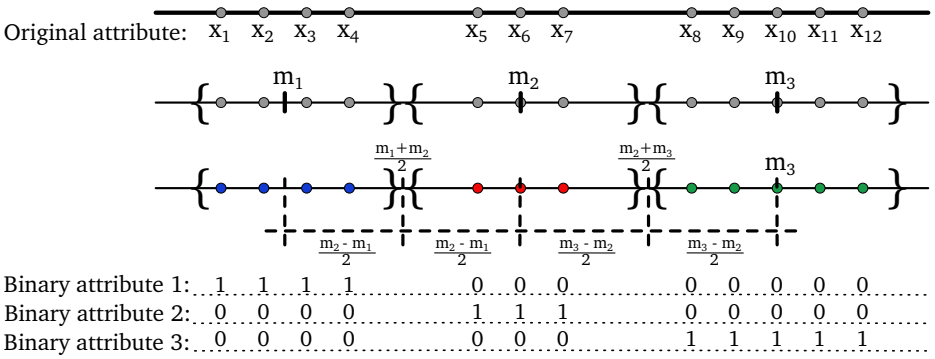


Figure 10: Graphical representation of the output of the procedure for finding the best partitioning of a given attribute.

Given k, attribute A and the best partitioning $\mathrm{P}_k(A)$ of the array A, we can create k new binary attributes according to the set A_j^s the value belongs to. The procedure of creating binary attributes for a given continuous attribute is illustrated in Figure 10.

Given a pair (a,b) and an array A, then a partitioning $\mathrm{P}_k(A)$ is called best on (a,b) if $H(\mathrm{P}_k(A)) \leq H(\mathrm{P}_j(A))$ for all $j = a, \cdots, b$. Such a partitioning into at least a and at most b arrays is denoted $\mathrm{P}_a^b(A)$. We introduce a procedure called *expand* that for each of the attributes A_i of a given dataset identifies the best partitioning $\mathrm{P}_2^9(A_i)$ (into at least two and at most 9 arrays) and then for each, such partitioning creates new binary attributes.

The binary attributes effectively slice the space into parts. Hence, the model that has binary attributes is a piecewise model. Just like trees use splits to divide the instance space, polynomials use binary attributes for a similar purpose, as shown in Figure 9.

4.2.2 Multi-target Polynomial Regression

The task of multi-target polynomial regression is to induce from data a polynomial equation that can predict the value of several variables.

The multi-target polynomial model is based on the general linear model[7]. It can be defined as:

$$P = \beta_0 + \sum_{i=1}^{m} \beta_i \cdot T_i \tag{24}$$

where $T_i = \prod_{j=1}^{n} X_j^{a_{i,j}}$ are the terms. Here $\beta_i,\ i = 1, \cdots, m$ and β_0 are constant vectors (not constants) and $\beta_i \neq \vec{0}$. The dimension of these vectors is the number of targets we want to predict.

An example of a multi-target polynomial equation is

$$P = (1,\ 2,\ 5) \cdot X_1^2 X_2 + (3,\ 5,\ 7) \cdot X_1 X_2^3 + (2,\ 3,\ 10).$$

This equation model predicts three targets. It is equivalent to the three single target polynomial equations given below, i.e., $P = (P_1, P_2, P_3)$, where

$$\begin{aligned} P_1 &= 1 \cdot X_1^2 X_2 + 3 \cdot X_1 X_2^3 + 2 \\ P_2 &= 2 \cdot X_1^2 X_2 + 5 \cdot X_1 X_2^3 + 3 \\ P_3 &= 5 \cdot X_1^2 X_2 + 7 \cdot X_1 X_2^3 + 10 \end{aligned}$$

The three equations P_1, P_2, and P_3 have same structure, but different coefficients for each target variable. For another consider Figure 11. The two functions $x^2 + 2x - 1$ and $-0.5x^2 - x + 1$ look very different but are actually very similar by structure.

[7] For short introduction of the general linear model, please see Section 2.2.2.

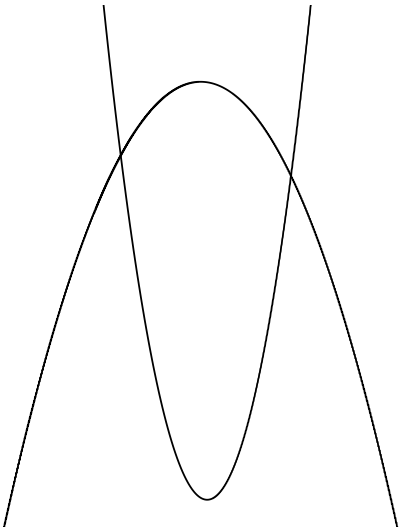


Figure 11: An example of two polynomial functions x^2+2x-1 and $-0.5x^2-x+1$, which look very different, but contain the same monomials and can be fitted simultaneously by using the multi-target version of Ciper.

In this way a single equation can be used for predicting several targets. The idea is that the complexity of this equation will be smaller than the complexity of a set of equations (one equation per each target). If the single target equations depend on the same terms, then we will need less information to encode the multi-target model than the tuple of single target models. In this case, we may obtain a model that has better predictive capabilities, because the risk of over-fitting will be smaller.

As already described in Section 4.1.2.3, by summing the stochastic complexities of the linear model for each target with the complexity of the structure we obtain the total complexity of the multi-target model.

The new Ciper algorithm for multi-target polynomial regression works just the same as in the single target case (Table 1). The data can be represented as a matrix X, where the number of rows is the number of instances, and the number of columns is the number of terms (m) plus one (the first column is filled with ones). We calculate the coefficients β_i of the equation as

$$\beta = (X^T \cdot X)^{-1} \cdot (X^T \cdot Y) \tag{25}$$

where Y is the matrix of values we are trying to predict.

We have introduced some optimizations of the calculation of the coefficients. 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 the matrices $X^T_{T_1,T_2} \cdot X_{T_1,T_2}$ and $X^T_{T_3,T_4} \cdot X_{T_3,T_4}$ are equal. We store all generated elements from the matrices $X^T \cdot X$. We use them later to calculate the matrices of the subsequently generated polynomials. Moreover, these matrices are the same for all targets. This optimization considerably lowers the amount of computation.

In the example in Figure 11, two functions, x^2+2x-1 and $-0.5x^2-x+1$, are shown. Instead of searching for two models, one for each target, the multi-target version of Ciper can find only one. Notice that, even thought they look very different, they still have the same structure. The multi-target version of Ciper will find the multi-target polynomial $(1,-0.5)x^2+(2,-1)x+(-1,1)$ as a single model for predicting both targets simultaneously.

4.2.3 Classification via Multi-Target Regression

The new Ciper , just as the old one, is designed to solve regression problems with continuous targets. In order to apply Ciper to solve discrete classification problems, we consider the

conditional class probability function and seek an approximation to it[8]. During classification, the class value whose model generates the greatest approximated probability value is chosen as the predicted class [17].

For each of the class values of the target attribute from the original dataset, we create a new dataset. Each derived dataset is the same as the original one, except for the class value which is set to 1 or 0 depending on whether that instance has the appropriate class or not. Next, a regression algorithm generates a model for each of the datasets. For a specific instance, the output of one of these models constitutes an approximation to the probability that this instance belongs to the associated class. An instance of unknown class is processed by each of the regression models. The class whose regression model gives the highest value is chosen as the predicted class. This process is illustrated in Figure 12.

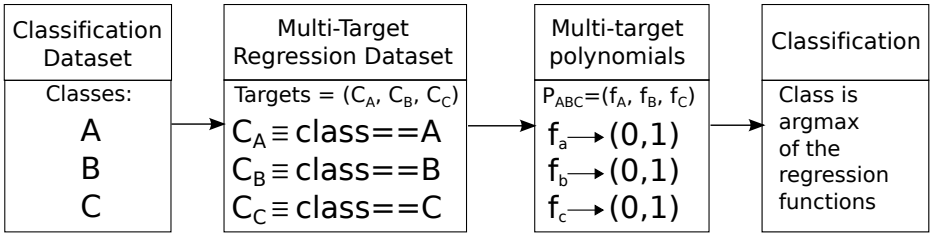


Figure 12: The process of classification via multi-target regression with Ciper.

In our case, we can use multiple runs of the Ciper algorithm for each of the binary targets, or, we can use the multi-target version of Ciper for a derived multi-target dataset which has the same attributes as the original dataset and the vector of all binary variables. If we consider the fact that the binary targets are semantically connected (as they were derived from the same desired target), it is reasonable to assume that the regression models should have similar structure. Therefor, the multi-target version of Ciper is more appropriate. Learning a single Ciper model as compared to learning several models, one for each target, considerably lowers the computational time. In some cases, the reduction factor can be larger than the number of targets. This is because of the semantic relationship that exists between the binary targets, and because of the decreased probability for over-fitting.

4.3 Implementation Details

The implementation of the new Ciper algorithm went through several cycles. The old version of the Ciper algorithm was implemented in C++ visual studio by Peter Ljubič. Using the specification described in Todorovski et al., we implemented a new version in java (2005). It was possible to use it within the WEKA data mining software toolkit [24]. A new version was then implemented in Python, and in Ruby (2006) as part of the IQ (Inducing Queries for Mining Patterns and Models) project. The Ruby version implemented an example of a framework for generating different kinds of models and combining them in order to generate better ones. Both the Python and the Ruby version included the new refinement operator and the new MDL heuristic. The Ruby version was the last to support the Ad-Hoc heuristic.

The version of Ciper was implemented in Free Pascal. It included assembler subroutines for using SSE processor instructions to speed up calculations (2007). Here SSE stands for Streaming SIMD Extensions and SIMD means Single instruction, multiple data. This version of Ciper was the first to support piecewise polynomial models, multiple targets and classification via regression.

[8]For short introduction of classification via regression, please see Section 2.5 and [17].

The final version was implemented in C++ (2007 - 2010). It was again based on a framework designed for the IQ project for generic mining of different types of models. We compiled the code with optimizations so that it uses vectorization.

The handmade assembler procedures from the Free Pascal version were automatically made for every for-cycle that can be optimized by using SSE instructions. The compiler directive produces code that unrolls the for-cycles and executes up to four floating point operations simultaneously, depending on the type of precision we are using (single or double). We also included *OpenMP* support, so that the software can use the multiple cores that the machine may have. This optimization unrolls the for-cycles into several threads and then executes every thread separately. For some applications, this can speed up the calculations several times.

The software has the following options:

```
train 'train.arff' [test 'test.arff']
[targets n1,n2,id1,id2,...] [attributes n1,n2,id1,id2,...]
[remove n1,n2,...,id1,id2,...] [mostImportantTargets n1,n2,id1,id2]
[initial 'polstr'] [maxvardeg 'num' (5)]
[maxDegree 'num' (9)] [maxSize 'num' (400)] [maxNonBinaryDegree 'num' (9)]
[noHeuristic mdl (mdl)] [expand]
[smallBeam largeBeam (largeBeam)]
[validationNumberOfFolds 'num' (10)] [classification]
[cv foldId numberOfFolds]
```

Ciper can read data in the arff file format, introduced by the WEKA data mining toolkit [24]. Ciper does not read discrete (nominal) attributes. For converting discrete attributes into numeric, we are using the WEKA software procedure *nominal to binary*. There are some Ciper related specifications relating the arff format. For example, we added the option to specify if the attribute is *binary*, *additive*, or *multiplicative*. The additive attributes will never be used for multiplication and the multiplicative attributes will be used only for multiplication with terms in the structure.

The next options concern the specification of the targets and the attributes. We have included important targets because sometimes we want to find models that best fit the important targets, and to avoid over-fitting we want some supporting targets. We can specify the attributes we want to use, and we can remove the attributes we don't want to use. The heuristic value for the important targets is multiplied by a coefficient *coefForMostImportantTargets* that has a default value of 4.

The *initial*, *maxvardeg*, *maxDegree*, and *maxSize* parameters are used to constrain the search space. *maxNonBinaryDegree* constrains the degree of the non-binary attributes for all the terms of the polynomial structure[9]. *Initial* specifies the initial polynomial structure. The *maxvardeg* specifies the maximum degree a variable can have; *maxDegree*, and *maxSize* specify the maximum degree and the maximum number of terms that the polynomial equation can have.

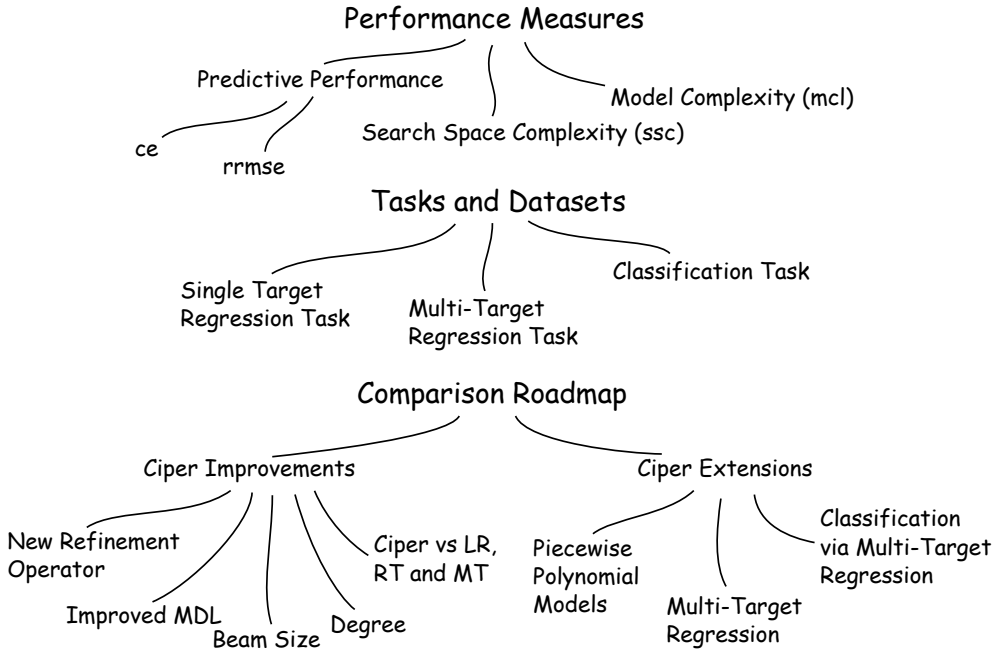
The *noHeuristic* option specifies the CV heuristic, and the *mdl* option the MDL heuristic. The *expand* option specifies whether we want piecewise models.The beam size option comes with two possibilities: *smallBeam* (size 8) and *largeBeam* (size 100). For the CV Heuristic we can choose the number of folds, with the *validationNumberOfFolds* option.

We can also specify that we want Ciper to solve a classification task with the *classification* option. For estimating the performance of Ciper using cross-validation we can use the *cv* option. This will generate results only on a specific singe fold from the cross-validation. We can use a small awk scripts to obtain the final results from the cross-validation. Since we are interested in comparisons with other algorithms, we used WEKA to split the datasets

[9]For more on maximum non-binary constraint, please see Section 7.1.

into train sets and tests sets using the method implemented there. In this way, we will use exactly the same folds by our algorithm and by other algorithms implemented WEKA.

5 Experimental Evaluation - Methodology



In this chapter, we will present the methodology used to evaluate the approaches presented in this thesis and to compare them to other existing approaches. We will first describe the measures used for evaluating the performance of the algorithms. We will then present the datasets on which we performed the evaluation and comparisons. Finally, we will present a roadmap of the evaluation and comparison process.

5.1 Performance Measures

We will analyze the performance of the new CIPER algorithm along three dimensions. The first dimension is the predictive performance of the models obtained with CIPER. The second dimension is the complexity of the search space explored with CIPER when solving a particular task. The third dimension is the size of the obtained polynomial models. With the analysis of the predictive performance, we will evaluate the predictive capabilities of the algorithm. With the analysis of the search space complexity, we will evaluate the computational demands of the algorithm. Finally, with the analysis of the model size, we will evaluate how complex the models are. This is also related to the interpretability of the models, since smaller models are easier to interpret.

5.1.1 Predictive Performance

For regression models, we will use the relative root mean squared error (*rrmse*) to measure the predictive performance. For single target models, the *rrmse* of a model P is calculated as follows:

$$\text{rrmse}(P) = \sqrt{\frac{\sum_{i=1}^{N}(P(x_i)-y_i)^2}{\sum_{i=1}^{N}(E(Y)-y_i)^2}} \tag{26}$$

For multi-target models, we look at the *rrmse* for each target separately.

For classification models, we consider the classification error:

$$\mathrm{ce}(P) = \frac{number\ of\ incorrectly\ classified\ instances\ with\ P}{total\ number\ of\ instances} \tag{27}$$

Classification error is in the range of $[0, 1]$, where 0 corresponds to the best performance.

To estimate the predictive performance of the obtained models on test data, unseen in the training phase, we follow the standard 10-fold cross-validation method. For a short introduction into cross-validation, please see Section 2.4.1. For a short introduction to model assessment and selection in general, please see Section 2.4. We use the acronyms *rrmse* and *ce* in the tables and graphs in Chapters 6 and 7 and in the Appendices A and B to refer to the predictive performance of the models measured in the way described above, estimated for unseen data by 10-fold cross-validation.

To test the statistical significance of the differences in performances between different algorithms on a single dataset, we apply a paired t-test to the results of the two algorithms on each of the folds of 10 fold cross-validation[1]. A paired t-test is a statistical hypothesis test that checks whether the difference between two responses measured on the same statistical unit has a mean value of zero. For example, suppose we measure the value of *rrmse* of an algorithm with and without some improvement. On a single dataset, we use the t-test to check whether the improved algorithm performs significantly better on the dataset at hand. We can also compare the algorithms on multiple datasets. If the improvement is effective, we expect that it will be significantly better on more datasets.

To test the statistical significance of the difference in performance between different algorithms on a number of datasets, we employ the Wilcoxon signed-rank test [67]. The Wilcoxon signed-rank test is a non-parametric statistical hypothesis test used to evaluate whether the means of two population samples are different. Like the paired t-test, the Wilcoxon signed-rank test is a paired difference test. It is used as an alternative to the paired t-test when the population can not be assumed to be normally distributed or when the data has ordinal scales.

An algorithm performs better than (or outperforms) another algorithm, if it leads to models with a significantly smaller value of *rrmse* on the datasets considered, the significance being measured by the considered statistical significance test with respect to the chosen significance level, i.e., p-value.

The p-value used for the statistical significance testing both for the paired t-test and for the Wilcoxon signed-rank test is 0.01. The p-value is the probability of obtaining a test statistic at least as extreme as the one that was observed. The null hypothesis is 'rejected' when the p-value is less than some significance level, in our case 0.01. Then the result is said to be statistically significant.

5.1.2 Search Space Complexity

We will measure the search space complexity with the number of equations that were generated by Ciper during the search. Given a dataset, the number of models generated for that dataset represents the complexity of the search space explored by the algorithm. We will refer to this number as *ssc*.

For a realistic estimate of this value, given the algorithm and a dataset, we use the values of *ssc* calculated on each of the 10 folds. We estimate it as the average of these values. We say that an algorithm performs better than another algorithm in terms of search space complexity, if it leads to models with a significantly smaller value of *ssc* on the datasets considered, the significance measured by the Wilcoxon signed-rank test.

[1]For short description of statistical comparisons of classifiers, see Section 2.4.4.

5.1.3 Model Complexity

We will compare estimates of the model complexity in terms of the following measures, introduced in Section 2.3:

- *Degree*, defined as the highest degree a term in the polynomial. The model is considered more complex if its degree is higher. This number is denoted by *mcd* and it is estimated as the average of the degrees of the polynomials on each fold.
- *Length*, defined as the sum of the degrees of all terms in the polynomial. The model is considered more complex if its length is higher. This number is denoted by *mcl* and it is estimated as the average of the lengths of the polynomials on each fold.
- *Size*, defined as the number of terms in the polynomial. The model is considered more complex if it has more terms. This number is denoted by *mcs* and it is estimated as the average of the sizes of the polynomials on each fold.

Note that we use an MDL measure of complexity to select polynomials during the learning phase. This is not to be confused with the measures discussed above. We consider these simplified measures of complexity of a model for the purpose of easily showing the difference between the models.

We say that an algorithm performs better than another algorithm in terms of model complexity, if it leads to models with a significantly smaller value of *mcl* on the datasets considered, the significance being measured by the Wilcoxon signed-rank test. When an algorithm performs better when measuring the length - *mcl*, then it usually performs better when measuring the other two measures (the degree - *mcd* and the size - *mcs*). This is the reason we only report *mcl* in the discussion of the results.

5.2 Tasks and Datasets

We used 33 regression datasets to evaluate the different approaches to the task of single target polynomial regression. Out of the 33 datasets, 10 datasets contain discrete (nominal) attributes. The list of datasets that do not contain discrete attributes, is given in Table 5. The list of datasets that contain discrete attributes is given in Table 6. On these 33 datasets, we evaluate the different Ciper improvements and extensions and compare their performance to the performance of other machine learning algorithms for regression.

We also used 10 multi-target regression datasets, listed in Table 7. These datasets were used to evaluate the capability of Ciper to solve the task of multi-target regression by learning polynomial models. In the context of this task, we also evaluate some of the improvements of Ciper that may influence the performance of the resulting models.

We tested our approach of classification via regression using Ciper on the 20 datasets in Table 8. We empirically evaluate several Ciper improvements on these datasets. Finally, we use them to compare our approach to other classification approaches.

All of the single target (classification and regression) datasets are publicly available. The classification datasets come from the *UCI Machine Learning Repository* [16]. The single-target regression datasets come from the UCI repository, as well as *Delve* [42], *StatLib* [64] and Luis Torgo dataset collections [62], discussed bellow.

The *UCI Machine Learning Repository* [16] is widely used by researchers as a primary source of machine learning datasets. It is one of the top 100 most cited references in all of computer science. We used both regression and classification datasets from the UCI repository. *Delve* [42] is another repository, designed for similar purposes as the UCI repository. It is an environment designed to evaluate the performance of many methods that learn relationships from empirical data. The datasets in *delve* are also used for comparing different learning methods. *StatLib* [64] is a system for distributing statistical software, datasets,

Table 5: The list of regression datasets that have numerical variables only. For each dataset, we give the acronym, its basic properties, and a reference to its source.

	Dataset	Instances (I)	Attributes (A)	Reference
01.	2dplanes	40768	11	[4]
02.	auto-price	159	16	[16]
03.	bank32nh	8129	33	[42]
04.	baskball	96	5	[56]
05.	bodyfat	252	15	[64]
06.	cal-housing	20640	9	[43]
07.	cpu-small	8192	13	[16]
08.	delta-ailerons	7129	6	[62]
09.	delta-elevators	9517	7	[62]
10.	elevators	16599	19	[62]
11.	elusage	55	3	[8]
12.	fried-delve	40768	11	[18]
13.	house-8l	22786	9	[42]
14.	housing	506	14	[16]
15.	mbagrade	61	3	[30]
16.	mv	40768	11	[62]
17.	pw-linear	200	11	[16]
18.	pyrim	74	28	[34]
19.	quake	2178	4	[56]
20.	triazines	186	61	[35]
21.	vineyard	52	4	[56]
22.	kin-8nm	8192	9	[42]
23.	puma32h	8192	33	[42]

and information by electronic mail, FTP and WWW. Another repository that we use is an online dataset archive made available by Luis Torgo [62].

Table 6: The list of regression datasets that also have discrete (nominal) variables. For each dataset, we give the acronym, its basic properties (number of instances, number of all and discrete attributes) and a reference to its source.

	Dataset	I	A	Discrete Attributes (N)	Reference
01.	cloud	194	7	2	[41]
02.	cholesterol	297	14	11	[16]
03.	fruitfly	125	5	2	[25]
04.	lowbwt	189	10	7	[16]
05.	meta	528	22	2	[2]
06.	pharynx	193	12	10	[16]
07.	sensory	576	12	11	[5]
08.	servo	167	5	4	[46]
09.	strike	625	7	1	[66]
10.	veteran	137	8	4	[31]

5.3 Comparison Roadmap

The evaluation of Ciper that we perform can be divided in three parts:

- *Single target regression.* Performed on the datasets from Tables 5 and 6. The purpose here is to evaluate the improvements of Ciper described in Section 4.1. Single target

Table 7: The list of multi-target regression datasets. For each dataset we give the acronym, its basic properties (number of instances, number of all and discrete (nominal) attributes, and the number of continuous target variables) and a reference to its source.

	Dataset	I	A	N	Targets	Reference
01.	edm	154	18	2	2	[32]
02.	solar-flare1	323	30	3	3	[16]
03.	solar-flare2	1066	30	3	3	[16]
04.	water-quality	1060	30	0	14	[14]
05.	sigmeareal	818	8	0	2	[11]
06.	habitat-hectares	16967	46	1	7	[36]
07.	soil-quality	1946	159	0	3	[10]
08.	prespa-lake	218	127	0	15	[37]
09.	soil-resilience	26	16	1	8	[10]
10.	sigmea-simulated	10369	13	2	2	[13]

Table 8: The list of classification datasets. For each dataset, we give its acronym, its basic properties (number of instances, number of attributes, and the number of targets, i.e., values of the class variable) as well as a reference to its source.

	Dataset	I	A	Targets	Reference
01.	mushroom	8124	23	2	[16]
02.	glass	214	10	6	[15]
03.	segment	2310	20	7	[16]
04.	vote	150	5	2	[52]
05.	anneal	898	25	5	[16]
06.	autos	193	25	6	[33]
07.	balance-scale	625	5	3	[55]
08.	cleveland-14	296	14	2	[16]
09.	german	1000	21	2	[16]
10.	heart-statlog	270	14	2	[16]
11.	hepatitis	118	18	2	[12]
12.	hungarian-14	261	11	2	[16]
13.	ionosphere	351	35	2	[16]
14.	lymphography	148	19	19	[16]
15.	pima-indians	768	9	2	[57]
16.	sick	3621	24	2	[16]
17.	sonar	208	61	2	[16]
18.	vehicle	846	19	4	[16]
19.	zoo	101	18	7	[16]
20.	vowel	990	14	11	[63]

regression is the basic task considered by Ciper and a starting point for addressing the other tasks.

- *Multi-target regression.* Performed on the datasets from Table 7. The purpose is to show that the multi-target version of Ciper is efficient and effective at solving this task and that it can be successfully used for solving multi-target problems. We can argue that it is even better to perform multi-target regression to solve some single target problems. By using attributes that we know are related to the target, we can

lower the probability of over-fitting and produce better models.

- *Classification.* Performed on the datasets from Table 8. Here we perform classification via multi-target regression by using Ciper. We evaluate most of the improvements of Ciper, but now with a focus on the classification task.

The first improvements we made to the original version of Ciper are the new refinement operator and the new improved MDL heuristic: we will start by evaluating them. In the first of the evaluations, Ciper uses the old AdHoc heuristic, and we focus on comparing the new refinement operator to the old one. Since we have significant improvements, we further use the better (new) refinement operator and focus on the heuristic. In the next step, Ciper with the new MDL heuristic is compared to Ciper with the old AdHoc heuristic. After this step, we already draw some conclusions on what Ciper options should be used for further comparisons.

We then investigate what happens when we modify the size of the beam, and what happens when we constrain the degree of the polynomials considered. Smaller beam sizes should reduce the execution time, i.e., lower the search space complexity. Constraining the degree will have a similar impact on the search space complexity, but will also influence the predictive performance of the learned models. We want to prove that polynomial equations have better predictive performance than linear and quadratic equations. It would be unreasonable to design an algorithm that learns polynomial equations if they perform worse. We are also interested in the performance of piecewise models. We are interested in the following questions about them: Do they yield any performance improvements, are there cases where they are useful, and because of the many new attributes introduced, do they increase the risk of over-fitting.

In the end, we will compare Ciper with other regression algorithms, such as linear regression, regression trees, and model trees. The purpose of this evaluation is to prove that Ciper, with the improvements we introduced, performs at least as well as most of the machine learning algorithms for the regression task.

We will then evaluate piecewise Ciper in the context of the regression classification task. The goal is to determine the effect that the use of piecewise models have on the performance of the algorithm. We will try to prove that piecewise models with a lower degree are as good as polynomial models. For this evaluation, we will need to constrain only the degree of the continuous variables. We will compare linear piecewise models, quadratic piecewise models and polynomial piecewise models with piecewise polynomial models.

The next step in the evaluation process addresses the task of multi-target regression. Here we will determine what heuristic is better for this task. The goal here is to set and evaluate Ciper as a multi-target algorithm that performs the task of regression for all targets specified. If there is a strong dependence among the targets, we even expect that Ciper will turn this into an advantage for building better models. We will also consider piecewise polynomial models in this context.

We finally turn to evaluating Ciper on the classification task. Because this task is different from regression, we will again evaluate the effect of constraining the degree of the learned polynomial models.We want to prove that polynomial equations have better predictive performance than linear and quadratic equations, also when considered in the context of classification. We will also consider piecewise polynomial models in this context.

We will then compare Ciper to other classification via regression algorithms. We will also compare it to other classification algorithms. The purpose is to show that Ciper performs this task as well as (if not better then) any other classification algorithm.

In the evaluation, we use many different versions of Ciper. The most commonly used versions will be referred to as follows:

- Old Ciper - The baseline, implemented with the old refinement operator and the old AdHoc heuristic.

- MDLCiper - Ciper with the new refinement operator and the new MDL heuristic.
- CVCiper - Ciper with the new refinement operator and the new CV heuristic.
- MDLCiperX - Piecewise Ciper with the new refinement operator and the MDL heuristic.
- CVCiperX - Piecewise Ciper with the new refinement operator and the CV heuristic.

For the purpose of comparing Ciper with other algorithms, we used the WEKA data mining toolkit [24]. All datasets, prior to the evaluation, were split into folds by using this software. In this way, the folds that the WEKA algorithms used and that Ciper used are exactly the same. The regression algorithms from WEKA that we used in our comparisons are: linear regression (LR), regression trees[2] (RT), and model trees[3] (MT).

WEKA implements a meta algorithm for classification via regression, that can be used with any regression algorithm in WEKA to perform the classification task. We compare Ciper to classification via linear regression (LR) and classification via model trees[4] (MT).

Finally, we compare Ciper with the following classification algorithms: decision trees[5] (J48); support vector machines[6] (SMO) with 3 different kernels: polynomial kernel with exponent 1 (SMO-E1), polynomial kernel with exponent 2 (SMO-E2) and a kernel with a radial basis function (SMO-KR); and Naive Bayes.

We will present the results in a step-wise fashion. We will use the best parameter settings for Ciper obtained so far in the subsequent comparisons. This means that we will always try to go from the worst performing parameter settings to best performing parameter settings. We will then use the best Ciper setting for the next round of evaluations.

[2]For a short introduction to regression trees, please see [4]
[3]For an introduction into model trees, please see [65]
[4]For more details, please see [17]
[5]For an introduction to decision trees, please see [45] and [44].
[6]For an introduction to support vector machines, please see [9].

6 Evaluating CIPER Improvements

In this chapter, we present the results of the empirical evaluation of the different CIPER improvements. Recall from Section 4.1 that these include the introduction of a new (improved) refinement operator and two search heuristics, one based on a new MDL encoding of polynomial models and the other based on cross-validated (CV) estimation of accuracy (error) on unseen data. Three sections report each of these aspects of evaluation and compare the effects of different CIPER improvement on its performance.

Two more sections consider the combined effects of the new search heuristics and the settings of two CIPER parameters. The parameters considered are beam size, i.e., the number of candidate equation structures kept at each stage of the search, and degree, i.e., the maximum degree that a polynomial induced by CIPER can have. The final section compares the two versions of CIPER (with the MDL and CV heuristics) and several standard regression algorithms (linear regression, regression trees and model trees).

The results of the evaluations are reported in this chapter in a condensed form, summarizing the results on two different collections of datasets, as explained below. The complete results can be found in Appendix A, which provides verbose tables with detailed results on each individual datasets from the collection. The summary results have the form of Tables and Figures, which have the structure and meaning described below.

Each summary table presents a statistical comparison of the performance of two versions of CIPER (and possibly some other regression algorithms) and follows a uniform structure. The first row in the table reports a summary of a per-dataset comparison of the difference in performance, i.e., the number of significant wins and losses of the first algorithm against the second algorithm. The significance is assessed according to the paired t-test, applied to the performance measure of the two algorithms on the 10 folds of 10-fold cross-validation, where a significance threshold of $p = 0.01$ is used. It is reported in the form of a $w : l$ score, where w and l denote the number of significant wins and loses for the first algorithm, respectively. A win for the first algorithm is a dataset where it has a significantly better performance for the metric considered (e.g., lower *rrmse*). The $w : l$ score is reported for all three performance measures, i.e., relative root mean square error (*rrmse*), the search space complexity (*ssc*) and model complexity (*mcl*): lower values are better for all three.

The second row in the table reports the significance of the difference in performance as measured by using the Wilcoxon signed-rank test. The outcome of the test is reported in the form of the p-value. Values larger than 0.99 or smaller than 0.01 denote a significantly different performance of the two algorithms: if $p > 0.99$ the first algorithm is better and if $p < 0.01$ the second algorithm is better.

Where possible, we also provide a visual comparison of the performance of the two algorithms on the individual datasets from the collection used for the evaluation. The graphs are structured as follows. The x-axis represents the *rrmse* while the y-axis represents the computational complexity (*sse*). Each circle or square corresponds to the performance of an algorithm applied to a dataset, blue-colored circles correspond to the performance of the first algorithm, while red-colored squares correspond to the performance of the second version. The centers of the square and the circle corresponding to the same dataset are connected with a line. The line is bold and blue if the first algorithm is significantly better than the

second in terms of *rrmse*, while it is bold and red if the second algorithm is significantly better than the first. If the line is thin and black, then the algorithms have similar prediction performances (which are not significantly different).

The position of the circles and squares simultaneously shows two aspects of the performance of an algorithm on a particular dataset. Smaller *rrmse* means they are positioned more towards the left, indicating better predictive performance. Smaller *search space complexity* means they are positioned lower, indicating more efficient search, i.e., lower computational complexity. The size of the circles and squares denotes the complexity of the obtained models. A point (small circle or square) placed in the lower left corner of the graph denotes optimal performance, where we deal with accurate and concise models obtained from a small search space.

The first two comparisons are related to the Ad-Hoc heuristic and the old refinement operator. They are implemented in an older version of Ciper, and are the first improvements that we introduced into Ciper[1]. For these two comparisons, we used only the datasets described in Table 5.

All the comparisons in this chapter involve single target regression. With the exception described above, we use all the regression datasets described in Tables 5 and 6. As already mentioned, for each comparison, we give tables with results in Appendix A.

6.1 Evaluating the New Refinement Operator

In this section, we compare the old Ciper refinement operator and the new improved refinement operator proposed in Section 4.1.1. The search heuristic that we use for this comparison is the old Ad-Hoc heuristic used in the old Ciper algorithm. For the comparison, we use the 23 single-target regression datasets with numeric attributes only, described in Table 5.

Ciper using the improved refinement operator generates models that have significantly better predictive performance than the models generated by Ciper using the old refinement operator. We say that Ciper using the improved refinement operator **performs better** than Ciper using the old refinement operator[2]. Ciper using the improved refinement operator generated models that have better predictive performance on 8 datasets. The outcome of the Wilcoxon signed-rank test indicates that the difference in performance between the corresponding versions of Ciper is statistically significant. This can be seen in Figure 13, where the blue circles are more to the left as compared to the red squares (especially those connected with a thick line, indicating a significant difference in performance for the specific dataset).

Table 9: A statistical comparison of the performance of Ciper using the new refinement operator and the old refinement operator, respectively. Summary of Table 45.

(new ref. op.) : (old ref. op.)	*rrmse*	*ssc*	*mcl*
t-test	8:0	0:23	3:15
w-test	0.9917	0.0	0.0295

Ciper using the old refinement operator searches fewer equations than Ciper using the improved refinement operator, for each of the 23 datasets. According to the Wilcoxon signed-rank test, this difference is statistically significant. This can be seen in Figure 13, where the red squares are placed lower than the blue circles.

Less complex models are generated by Ciper using the old refinement operator than by Ciper using the improved refinement operator. While Ciper using the improved refinement

[1]For more on the implementation details of Ciper, please see Section 4.3.

[2]The predictive performance measures we use are defined and discussed in Section 5.1.1.

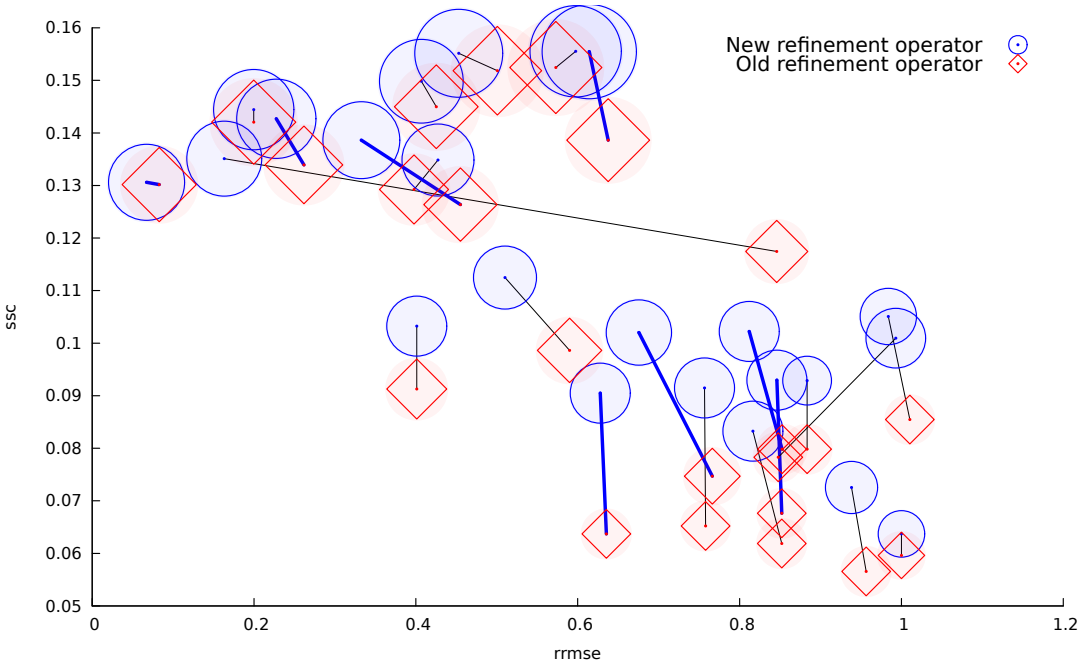


Figure 13: A visual comparison of the performance of Ciper using the new refinement operator and the old refinement operator, respectively.

operator generates models that have smaller complexity on 3 datasets, Ciper using the old refinement operator generates models that have smaller complexity on 15 datasets. According to the Wilcoxon signed-rank test, the difference in the size of the models generated by the two approaches is statistically significant. This can be seen in Figure 13, where the red squares are smaller than the blue circles.

With the new refinement operator, the refinement of an equation structure will always have a smaller error (on the training data). The complexity of the equations increases faster with the new than with the old refinement operator. As more complex equations have more refinements, the branching factor of the refinement graph is larger for the new operator. Hence, a larger number of equation structures is considered with the new refinement operator.

The use of the new refinement operator improves the predictive performance of Ciper. The search space is enlarged and more models are explored when using the new refinement operator. The resulting models are more complex, but also more accurate. We can thus conclude that it is better to use the new refinement operator than the old refinement operator. We will use only the new improved refinement operator in the comparisons that follow.

6.2 Evaluating the Improved MDL Heuristic

In this section, we compare the old Ciper Ad-Hoc heuristic and the new improved MDL heuristic proposed in Section 4.1.2. For both, the refinement operator in use is the new improved refinement operator described in Section 4.1.1. For the comparison, we use the 23 single-target regression datasets with numeric attributes only, described in Table 5.

Ciper using the improved MDL heuristic performs better than Ciper using the Ad-Hoc heuristic. Ciper using the improved MDL heuristic performs significantly better on 8 datasets. According to the Wilcoxon signed-rank test, the difference is statistically significant. This can be seen in Figure 14, where the blue circles are mostly to left of the corresponding red squares.

Table 10: A statistical comparison of the performance of CIPER using the new improved MDL heuristic and the old Ad-Hoc heuristic, respectively. Summary of Table 46.

(improved MDL) : (Ad-Hoc)	*rrmse*	*ssc*	*mcl*
t-test	8:0	17:5	21:2
w-test	0.9407	0.9932	0.9997

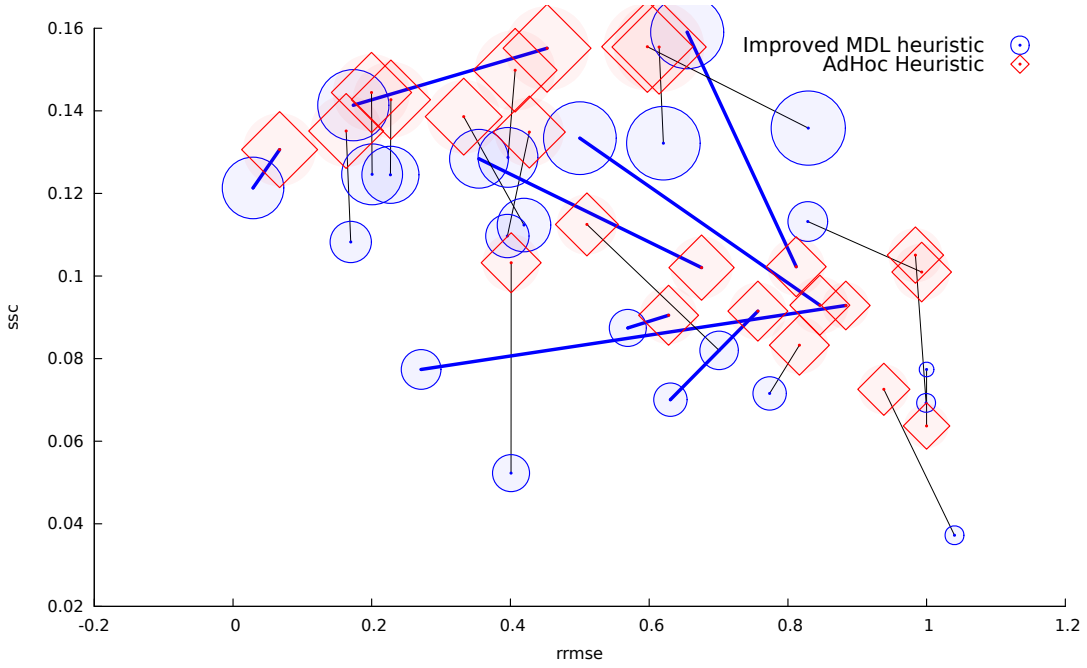


Figure 14: A visual comparison of the performance of CIPER using the new improved MDL and the old Ad-Hoc heuristic, respectively.

CIPER using the improved MDL heuristic searches fewer equations than CIPER using the Ad-Hoc heuristic. This can be seen in Figure 14: The blue circles are placed lower than the corresponding red squares. Less complex models are generated by CIPER using the improved MDL heuristic. This can also be seen in Figure 14, where the blue circles are smaller than the red squares.

Not only does the MDL heuristic improve the predictive performance of CIPER: Fewer models are explored when using it. The resulting models are also less complex. We can conclude that it is better to use the new improved MDL heuristic than the old Ad-Hoc heuristic. For further comparisons, we will not use the Ad-Hoc heuristic function.

The first two improvements, the improved refinement operator and the improved MDL heuristic produce better models in the sense of prediction accuracy, complexity of the models, and search space complexity. We will now consider the use of the CV heuristic, proposed in Section 4.1.3.

6.3 Comparing the Two New Search Heuristics: CV vs MDL

In this section, we compare the CV heuristic proposed in Section 4.1.3 and the MDL Heuristic proposed in Section 4.1.2. CIPER that uses the MDL heuristic will be referred to as MDLCIPER, while CIPER that uses the CV heuristic will be referred to as CVCIPER. MDLCIPER and CVCIPER will be applied to the 33 single-target regression datasets described in Tables 5 and 6.

MDLCiper (Ciper with the MDL heuristic) and CVCiper (Ciper with the CV heuristic) have similar predictive performance. CVCiper performs significantly better than MDLCiper on 4 datasets and worse on 4 datasets. The outcome of the Wilcoxon signed-rank test suggests that the difference in performance is not statistically significant.

Table 11: A statistical comparison of the performance of CVCiper and MDLCiper. Summary of Table 47.

CVCiper : MDLCiper	*rrmse*	*ssc*	*mcl*
t-test	4:4	0:31	0:31
w-test	0.5141	0.0	0.0

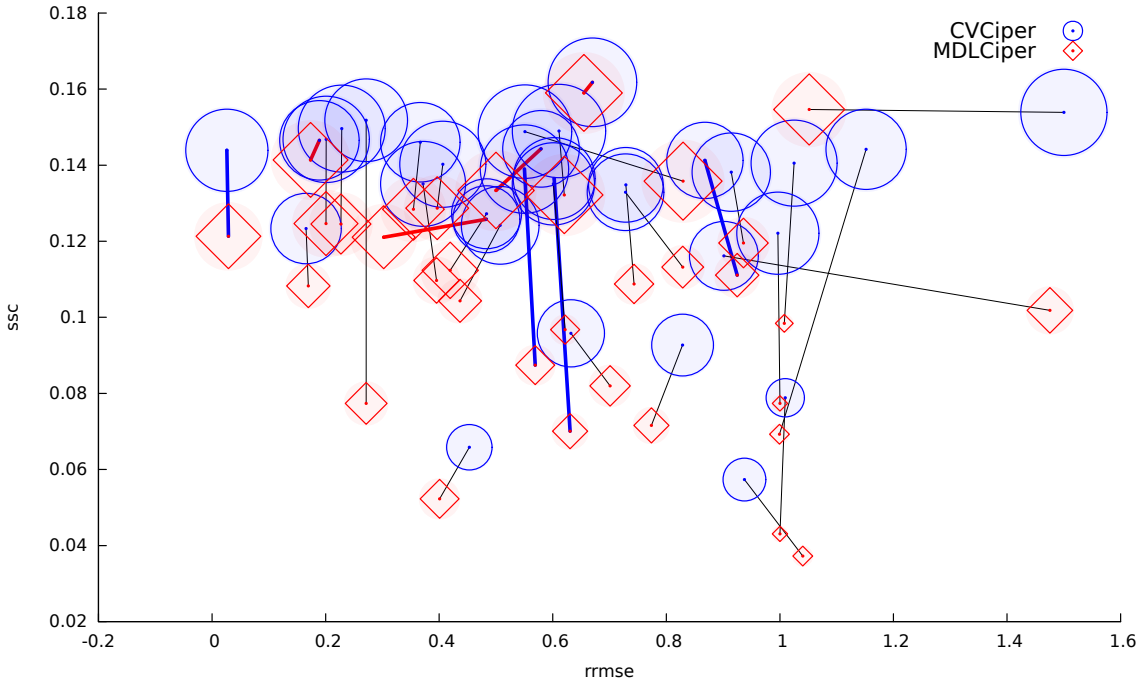


Figure 15: A visual comparison of the performance of CVCiper and MDLCiper, respectively.

MDLCiper searches a smaller space of equations as compared to CVCiper. This can be seen in Figure 15, where the red squares are placed lower than the corresponding blue circles. Less complex models are generated by MDLCiper than by CVCiper. This can also be seen in Figure 15, where the red squares are smaller than the blue circles.

We can conclude that MDLCiper and CVCiper have similar predictive performance. Given that MDLCiper explores fewer models and generates simpler models, we can say that MDLCiper is the better of the two for the single target regression task.

6.4 Evaluating the Effect of Beam Size

In this section, we evaluate the effect the beam size has on the performance of the algorithm. We do this by comparing Ciper using a small beam size (8) and Ciper using a large beam size (100). In both, the refinement operator used is the new improved refinement operator described in Section 4.1.1. The heuristics used are the improved MDL heuristic described in Section 4.1.2 and the new CV heuristic described in Section 4.1.3. The comparison uses the 33 single-target regression datasets described in Tables 5 and 6.

CIPER using the improved MDL heuristic

CIPER using a small beam and CIPER using a large beam have similar predictive performance. CIPER using a large beam is significantly better on 5 datasets and worse on 2 datasets. According to the Wilcoxon signed-rank test, the difference between the two beam sizes is not statistically significant.

Table 12: A statistical comparison of CIPER using the improved MDL heuristic with a large beam and a small beam, respectively. Summary of Table 48.

(large beam) : (small beam)	*rrmse*	*ssc*	*mcl*
t-test	5:2	1:25	4:12
w-test	0.8969	0.0	0.0057

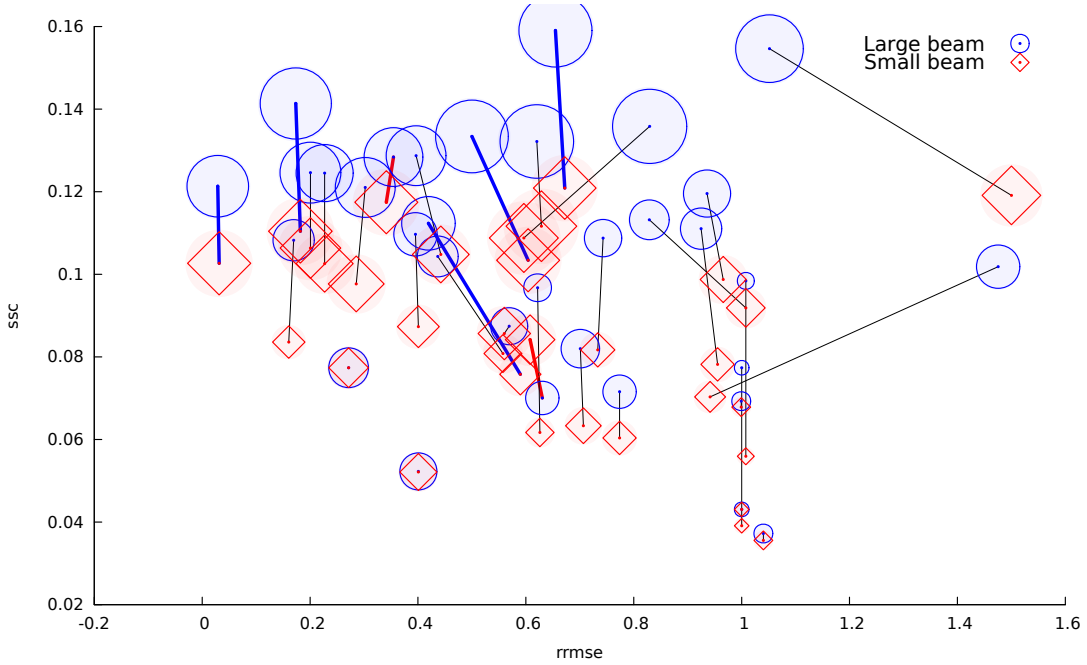


Figure 16: A visual comparison of CIPER using the improved MDL heuristic with a large beam and a small beam, respectively.

CIPER using a small beam searches fewer equations than CIPER using a large beam. This can be seen in Figure 16, where the red squares are placed lower than the blue circles. Less complex models are generated with CIPER by using a small beam than by using a large beam, as can also be seen in Figure 16: the red squares are smaller than the blue circles.

CIPER using the CV heuristic

CIPER using a large beam performs better than CIPER using a small beam, being significantly better on 13 datasets. The Wilcoxon signed-rank test indicates that the difference between using a large and a small beam is statistically significant. This can be seen in Figure 17, where the blue circles are mostly on the left of the corresponding red squares.

CIPER using a small beam considers fewer equations, as can be seen in Figure 17, where the red squares are placed lower than the blue circles. Simpler models are generated with CIPER by using a small beam than by using with a large beam. This can also be seen in Figure 17, where the red squares are smaller than the blue circles.

Table 13: A statistical comparison of CIPER using the CV heuristic, with a large beam and a small beam, respectively. Summary of Table 49.

(large beam) : (small beam)	*rrmse*	*ssc*	*mcl*
t-test	13:1	0:30	0:30
w-test	0.9997	0.0	0.0

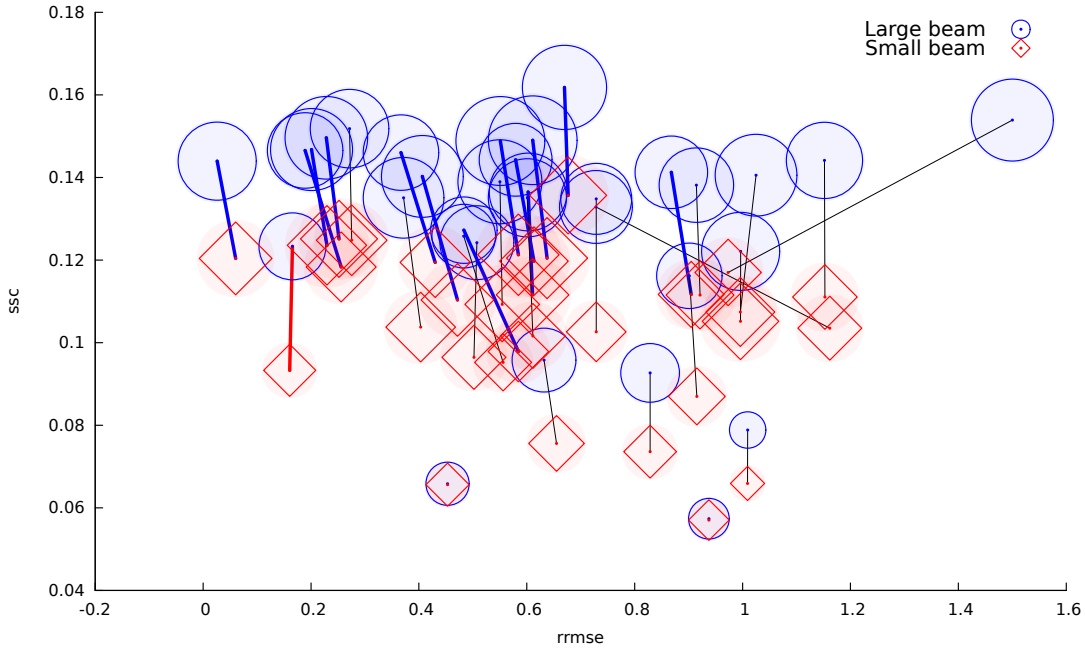


Figure 17: A visual comparison of CIPER using the CV heuristic with a large beam and a small beam, respectively.

The use of a large beam improves the predictive performance of the resulting models. More models are explored when using a large beam size and the resulting models are more complex. We can conclude that it is better to use a large beam than a small one if we place the emphasis on predictive performance and are willing to accept the extra computational and model complexity.

6.5 Evaluating the Effect of Degree

In this section, we evaluate the effect that the maximum allowed degree of the considered polynomials has on the performance of the CIPER algorithm. We do this in two steps. We first compare quadratic equations (learned by CIPER with a limitation of the maximum degree to 2) and linear equations (learned by CIPER with a limitation of the maximum degree to 1). We then compare polynomial equations (learned by CIPER with no limitation on the degree) and quadratic equations. The goal is to empirically prove that modeling with polynomial equations is better than modeling with linear or quadratic equations. The refinement operator used is the new improved refinement operator described in Section 4.1.1. The heuristic functions used are the improved MDL heuristic described in Section 4.1.2 and the CV heuristic described in Section 4.1.3. The comparison uses the 33 single-target regression datasets described in Tables 5 and 6.

CIPER using the MDL heuristic: Quadratic vs Linear Equations

The algorithm that generates quadratic equations (CIPER with a limitation of the maximum degree to 2) performs better than the algorithm that generates linear equations (CIPER with a limitation of the maximum degree to 1). The algorithm that generates quadratic equations performs significantly better on 8 datasets. The Wilcoxon signed-rank test indicates that the difference between quadratic and linear equations is statistically significant. This can be seen in Figure 18, where the blue circles are to the left of the corresponding red squares.

Table 14: A statistical comparison of CIPER using the improved MDL heuristic to learn quadratic and linear equations, respectively. Summary of Table 50.

quadratic : linear	*rrmse*	*ssc*	*mcl*
t-test	8:0	0:22	1:15
w-test	0.9963	0.0	0.0

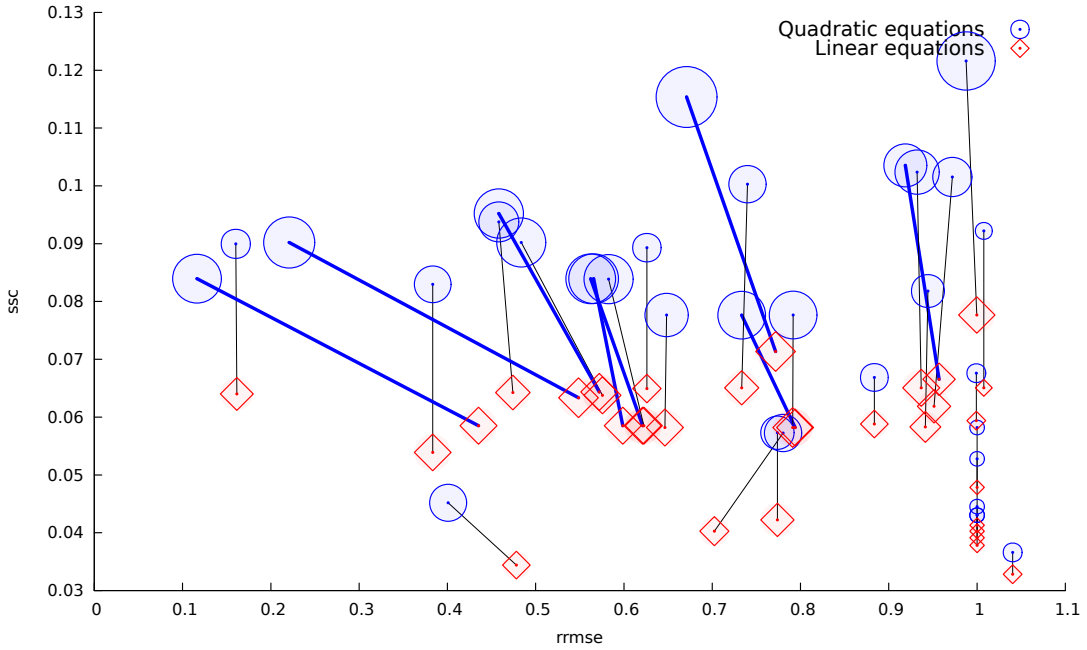


Figure 18: A visual comparison of CIPER using the improved MDL heuristic to learn quadratic and linear equations, respectively.

The algorithm that generates linear equations searches fewer equations than the algorithm that generates quadratic equations. This can be seen in Figure 18, where the red squares are closer to the bottom than the blue circles. Simpler models are generated by the algorithm that generates linear equations, as can be seen in Figure 18, where the red squares are smaller than the blue circles.

CIPER using the CV Heuristic: Quadratic vs Linear Equations

The algorithm that generates quadratic equations performs better than the algorithm that generates linear equations, being significantly better on 15 datasets and worse on 2 datasets. According to the Wilcoxon signed-rank test, the difference in performance between quadratic and linear equations is statistically significant. This can be seen in Figure 19, where the blue circles are to the left of the red squares.

Table 15: A statistical comparison of CIPER using the CV heuristic to learn quadratic and linear equations, respectively. Summary of Table 51.

quadratic : linear	*rrmse*	*ssc*	*mcl*
t-test	15:2	0:32	0:32
w-test	1.0	0.0	0.0

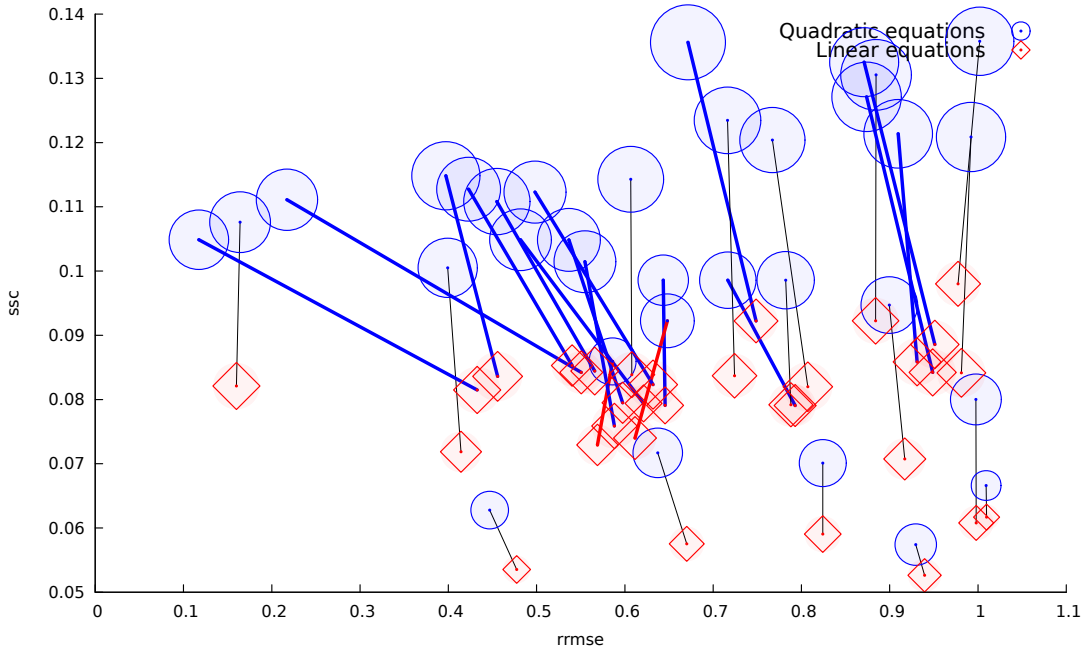


Figure 19: A visual comparison of CIPER using the CV heuristic to learn quadratic equations and linear equations, respectively.

The algorithm that generates linear equations searches fewer equations than the algorithm that generates quadratic equations. This can be seen in Figure 19, where the red squares are placed lower than the blue circles. Simpler models are generated by the algorithm that generates linear equations, as can also be seen in Figure 19, where the red squares are smaller than the blue circles.

CIPER using MDL Heuristic: Polynomial vs Quadratic Equations

The algorithm that generates polynomial equations (CIPER with no limitation on the degree) performs better than the algorithm that generates quadratic equations, being significantly better on 15 datasets. The Wilcoxon signed-rank test indicates the difference between polynomial and quadratic equations is statistically significant. This can be seen in Figure 20, where the blue circles are to the left of the red squares.

Table 16: A statistical comparison of CIPER using the improved MDL heuristic to learn polynomial and quadratic equations, respectively. Summary of Table 52.

polynomial : quadratic	*rrmse*	*ssc*	*mcl*
t-test	15:0	0:25	0:14
w-test	0.9959	0.0	0.0

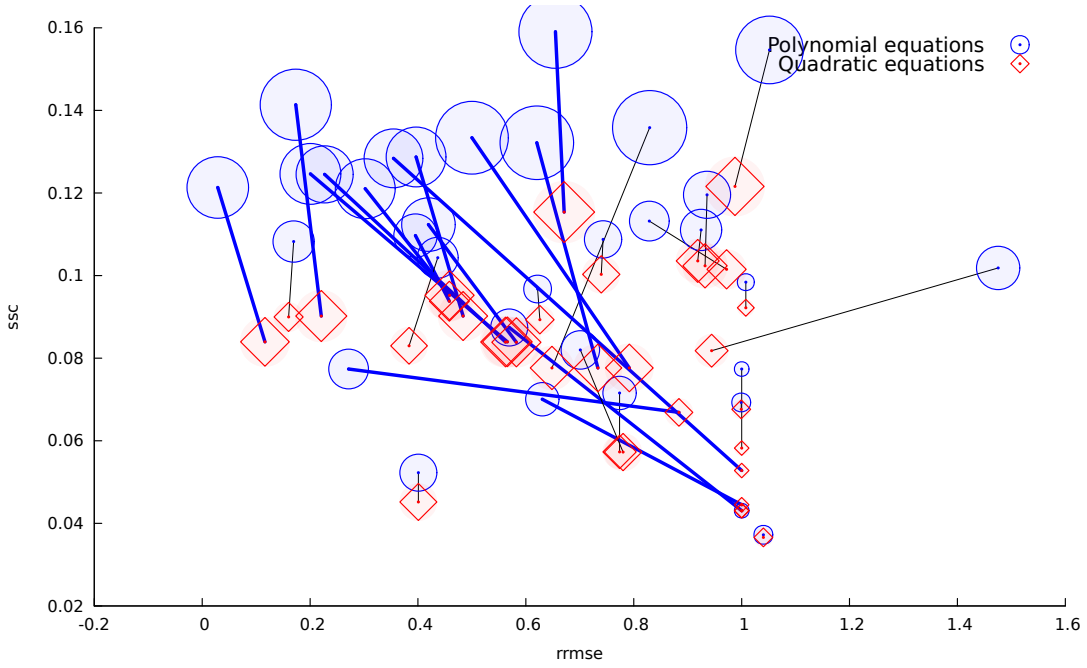


Figure 20: A visual comparison of CIPER using the improved MDL heuristic to learn polynomial and quadratic equations, respectively.

The algorithm that generates quadratic equations searches fewer equations, as can be seen in Figure 20, where the red squares are placed lower than the blue circles. The quadratic models are simpler. This can be also seen in Figure 20, where the red squares are smaller than the blue circles.

CIPER using the CV Heuristic: Polynomial vs Quadratic Equations

The algorithm that generates polynomial equations performs better than the algorithm that generates quadratic equations, being significantly better on 12 datasets. According to the Wilcoxon signed-rank test, the difference in performance between polynomial and quadratic equations is statistically significant. This can be seen in Figure 21, where the blue circles are to the left of the red circles.

Table 17: A statistical comparison of CIPER using the CV heuristic to learn polynomial and quadratic equations, respectively. Summary of Table 53.

polynomial : quadratic	*rrmse*	*ssc*	*mcl*
t-test	12:0	0:31	0:28
w-test	0.9831	0.0	0.0

The algorithm that generates quadratic equations searches fewer equations than the algorithm that generates polynomial equations. This can be seen in Figure 21, where the red squares are placed lower than the blue circles. The quadratic equations are simpler, as can also be seen in Figure 21, where the red squares are smaller than the blue circles.

Quadratic equations improve the predictive performance of CIPER as compared to linear equations, and polynomial equations improve the predictive performance of CIPER as compared to quadratic equations. More models are explored when learning quadratic equations than when learning linear equations. The resulting models are also more complex. With a focus on predictive performance, we can conclude that it is better to learn polynomial equa-

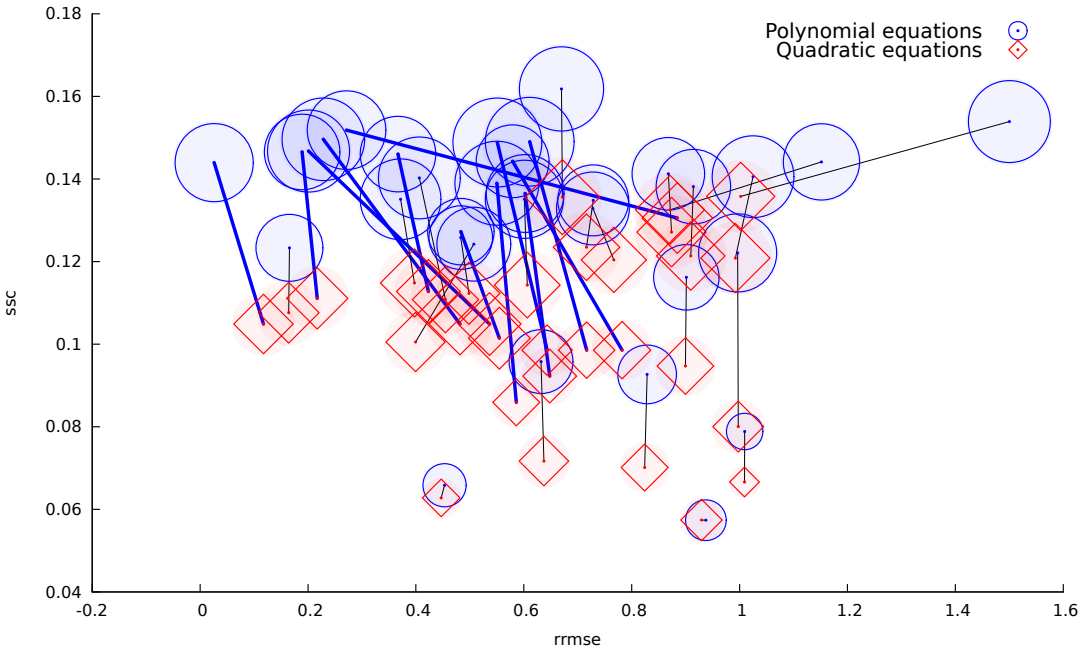


Figure 21: A visual comparison of Ciper using the CV heuristic to learn polynomial equations and quadratic equations, respectively.

tions than quadratic and linear ones, and it is better to learn quadratic equations than linear ones. Linear and quadratic equations are not always enough. In most cases, polynomials provide a reasonable and powerful extension that (as we have shown empirically) provides better models in terms of predictive accuracy. For future comparisons (Section 6.6), Ciper will learn polynomial equations without a limitation on the degree.

6.6 Ciper vs LR, RT, and MT

In this section, we compare MDLCiper and CVCiper with other machine learning algorithms for regression as described in Section 5.3. The comparison is made on the 33 single-target regression datasets described in Tables 5 and 6. The regression algorithms are implemented in the WEKA data mining toolkit [24]. All datasets, prior to evaluation, were split into folds using this software. In this way, the WEKA algorithms and our algorithm used exactly the same folds. The regression algorithms from WEKA that we used in the comparison are: linear regression (LR), regression trees[3] (RT), and model trees[4] (MT).

Table 18: Comparison of MDLCiper to LR, RT, and MT in terms of predictive performance. Summary of Table 54.

MDLCiper :	*LR*	*RT*	*MT*
t-test	11:2	13:1	3:3
w-test	0.9674	0.9974	0.148

Table 19: Comparison of CVCiper vs LR, RT, and MT in terms of predictive performance. Summary of Table 55.

CVCiper :	*LR*	*RT*	*MT*
t-test	12:0	13:1	3:4
w-test	0.9969	0.9999	0.1279

Tables 18 and 19 compare the performance of MDLCiper (respectively CVCiper). MDLCiper performs better than linear regression, being significantly better on 11 datasets and worse on 2 datasets. The Wilcoxon signed-rank test indicates that the difference between

[3]For short introduction about regression trees, please see [4]
[4]For short introduction about model trees, please see [65]

MDLCIPER and linear regression is statistically significant. MDLCIPER performs better than regression trees, being significantly better on 13 datasets and worse on 1 dataset. According to the Wilcoxon signed-rank test, the difference between MDLCIPER and regression trees is statistically significant. MDLCIPER and model trees have similar predictive performance, MDLCIPER being significantly better on 3 datasets and worse on 3 datasets. The Wilcoxon signed-rank test indicates that the difference between MDLCIPER and model trees is not statistically significant.

CVCIPER performs better than linear regression, being significantly better on 12 datasets. According to the Wilcoxon signed-rank test, the difference between CVCIPER and linear regression is statistically significant. CVCIPER performs better than regression trees, being significantly better on 12 datasets and worse on 1 dataset. According to the Wilcoxon signed-rank test, the difference between CVCIPER and regression trees is statistically significant. CVCIPER and model trees have similar predictive performance, CVCIPER performs better on 3 datasets and model trees on 4 datasets. Overall, the difference is not statistically significant.

We can conclude that MDLCIPER and CVCIPER perform better than linear regression and regression trees. As established in Section 6.3, they have similar predictive performance when comparing between themselves. They also have similar predictive performance as compared to model trees.

6.7 Summary

This section summarizes the results presented in the previous sections of this chapter.

We first evaluated (Section 6.1) the new improved CIPER refinement operator, comparing it to the old one. In this evaluation, we used the old Ad-Hoc search heuristic from the old CIPER algorithm. The new refinement operator improved the predictive performance of CIPER. However, more models are explored when using the new refinement operator and the resulting models are more complex. With an emphasis on predictive performance, we concluded that it is better to use the new refinement operator and used it in all further comparisons/evaluations.

The old CIPER Ad-Hoc heuristic and the new improved MDL heuristic were compared next (Section 6.2). The new refinement operator was used for this comparison. The use of the new MDL heuristic improved the predictive performance of CIPER. Furthermore, fewer models are explored when using the new MDL heuristic and the resulting models are simpler. For each of the three metrics that we use, the decision is unanimous: it is better to use the improved MDL heuristic than the Ad-Hoc heuristic.

We next consider an additional search heuristic, based on a cross-validation estimate of the error of the polynomial equations. In Section 6.3, we compared CVCIPER and MDLCIPER (using the CV and MDL heuristics, respectively). MDLCIPER and CVCIPER have similar predictive performance. However, MDLCIPER explores fewer candidate models and generates simpler models.

We then investigated (Section6.4) the effect that the beam size has on the performance of CIPER. We considered a small beam size (8) and a large beam size (100) and compared the performance of CIPER using each of these. These settings were used in conjunction with each of the new heuristics, MDL and CV. The use of a large beam improved the predictive performance of CIPER, especially when using the CV heuristic. However, more candidate models are explored when using a large beam size and the generated polynomials are more complex. With an emphasis on predictive performance, we concluded that it is better to use a large beam size: We use this in all comparisons, except for those in Section 6.4.

We further evaluated (Section 6.5) the effect that the maximum allowed degree of the learned polynomial models has on the performance of CIPER. We first compared CIPER learning quadratic and linear equations, then CIPER learning polynomial and quadratic

equations. Quadratic equations learned by Ciper performed better than linear equations and polynomial equations performed best. The number of models explored and the complexity of the resulting models increased with the performance. With an emphasis on predictive performance, we concluded that it is better to use polynomial than quadratic (and quadratic than linear) equations. The development of better machine learning algorithms that generate polynomial equations, the major goal of the present thesis, is thus appropriately justified.

Finally (Section 6.6), we compared MDLCiper and CVCiper with other standard machine learning algorithms for regression. These included linear regression, regression trees, and model trees. The results of the comparison show that MDLCiper and CVCiper perform better than linear regression and regression trees and have predictive performance similar to that of model trees.

7 Evaluating CIPER Extensions

In this chapter, we present the results of the empirical evaluation of the different CIPER extensions. Recall from Section 4.2 that these include the learning of piecewise polynomial models, the learning of multi-target polynomial models, and the use of polynomial models in the context of classification via regresssion. The first three sections of this chapter report on the results of the evaluation of the performance of each of these extensions. The last section summarizes the findings of these evaluation efforts.

We first investigate the performance of piecewise polynomial models as compared to ordinary polynomial models. In this context, we also consider the influence of the maximum degree that a polynomial induced by CIPER can have and the influence of the search heuristic used within CIPER (MDL or CV). We also compare the performance of the piecewise polynomial models learned by CIPER to the performance of models learned by several standard regression algorithms (linear regression, regression trees and model trees). The comparisons in this section are performed on the 33 single-target regression datasets described in Tables 5 and 6.

We then turn to the task of multi-target regression. Here we investigate the relative performance of ordinary and piecewise models. We also consider the influence of the search heuristic (MDL or CV). The comparisons in this section are performed on the 10 multi-target regression datasets described in Table 7.

We finally consider the task of classification, solving it with the approach of classification via multi-target regression. In this context, we use both ordinary and piecewise polynomial models. Orthogonally to this, we study the influence of the maximum degree that a polynomial can have and the influence of the search heuristic used within CIPER (MDL or CV). We also compare classification via regression with CIPER to classification via several standard regression algorithms (linear regression, regression trees and model trees), as well as to several standard classification algorithms (decision trees, support vector machines, naive Bayes). The comparisons in this section are performed on the 20 classification datasets described in Table 8.

The results of the evaluations are reported in this chapter in a condensed form, summarizing the results on different collections of datasets, as explained above. The complete results can be found in Appendix B, which provides verbose tables with detailed results on each individual dataset from the collection at hand. The summary results have the form of Tables and Figures, which have the structure and meaning described below.

Each summary table presents a statistical comparison of the performance of two versions of CIPER (and possibly some other regression algorithms) and follows a uniform structure. The first row in the table reports a summary of a per-dataset comparison of the difference in performance, i.e., the number of significant wins and losses of the first algorithm against the second algorithm. The significance is assessed according to the paired t-test, applied to the performance measure of the two algorithms on the 10 folds of 10-fold cross-validation, where a significance threshold of $p = 0.01$ is used. It is reported in the form of a $w : l$ score, where w and l denote the number of significant wins and loses for the first algorithm, respectively. A win for the first algorithm is a dataset where it has a significantly better performance for the metric considered (e.g., lower *rrmse*). The $w : l$ score is reported for three performance

measures, i.e., the predictive error, the search space complexity (*ssc*) and model complexity (*mcl*): lower values are better for all three. Predictive error is measured by the relative root mean square error (*rrmse*) for regression and by classification error (*ce*) for classification.

The second row in the table reports the significance of the difference in performance as measured by using the Wilcoxon signed-rank test. The outcome of the test is reported in the form of the p-value. Values larger than 0.99 or smaller than 0.01 denote a significantly different performance of the two algorithms: if $p > 0.99$ the first algorithm is better and if $p < 0.01$ the second algorithm is better.

Where possible, we also provide a visual comparison of the performance of the two algorithms on the individual datasets from the collection used for the evaluation. The graphs are structured as follows. The x-axis represents the *rrmse* (or the *ce*, depending on the task), while the y-axis represents the computational complexity (*sse*). Each circle or square corresponds to the performance of an algorithm in predicting a single target from a given dataset. Blue-colored circles correspond to the first algorithm, while red-colored squares correspond to the second algorithm.

The centers of the square and the circle corresponding to the same dataset/target are connected with a line. The line is bold and blue if the first algorithm is significantly better than the second in terms of *rrmse*, while it is bold and red if the second algorithm is significantly better than the first. If the line is thin and black, then the algorithms have similar prediction performances (which are not significantly different).

The position of the circles and squares simultaneously shows two aspects of the performance of an algorithm on a particular dataset. Smaller *rrmse*/*ce* means they are positioned more towards the left, indicating better predictive performance. Smaller *search space complexity* means they are positioned lower, indicating more efficient search, i.e., lower computational complexity. The size of the circles and squares denotes the complexity of the obtained models. A point (small circle or square) placed in the lower left corner of the graph denotes optimal performance, where we deal with accurate and concise models obtained from a small search space.

For single target datasets, we have one circle/square per dataset, while for multi-target datasets, we have as many circles/squares as the dataset has targets. Since a single model is used for predicting all targets, all the circles/squares have the same size. Also, since the same search space is explored, they have the same value for *ssc* and are positioned at the same height.

The statistical significance tests for multi-target algorithms and datasets are performed as follows. The paired t-test is applied to all pairs of performance figures for the two compared approaches on the dataset at hand, resulting from 10-fold cross-validation, for all 10 folds and all targets, taken together. The Wilcoxon signed-rank test comparing the performance of two approaches on a collection of multi-target datasets is applied to the set of all pairs of cross-validated estimates of *rrmse*, for each target and each dataset, taken together.

7.1 Evaluating Piecewise Polynomial Models

In this section, we evaluate the performance of CIPER learning piecewise polynomial models. Our goal is to prove that piecewise polynomial models with lower degrees can perform as well as general polynomial models (of arbitrary degree). In order to limit the degree of the polynomial, but allow for arbitrary partitions within the piecewise models, we introduce a constraint (called maximum non-binary degree) that allows terms in the polynomial to have arbitrarily many binary variables, but limits the degree on the part of the term containing normal variables. Let $x_1, x_2, \cdots, x_{n_a}$ be the non-binary attributes and let $z_1, z_2, \cdots, z_{n_b}$ be the binary attributes. A term in the polynomial structure is $x_1^{a_1} \cdot x_2^{a_2} \cdots x_{n_a}^{a_{n_a}} \cdot z_1^{b_1} \cdot z_2^{b_2} \cdots z_{n_b}^{b_{n_b}}$ and a

limitation of *maximum non-binary degree* L means that $\sum_{i=1}^{n_a} a_i \leq L$. We can have indefinitely many binary multipliers but the degree of the non-binary multipliers is limited. In this way, we can test the effect the piecewise models have compared to normal polynomial models.

The comparison will be performed in several steps: we will compare polynomial models and piecewise linear models (polynomial models with a maximum non-binary degree constrained to 1), then polynomial models and piecewise quadratic models (polynomial models with a maximum non-binary degree constrained to 2), and finally polynomial models and piecewise polynomial models without degree constraints. We will also compare the performance of the piecewise polynomial models learned by the MDL and CV versions of Ciper. In the end, we will compare piecewise polynomial models learned by the two versions of Ciper with the models learned by other regression algorithms.

The piecewise version of MDLCiper is labeled MDLCiperX, and the piecewise version of CVCiper is labeled CVCiperX. All comparisons in this Section 7.1 are performed on the 33 single-target regression datasets described in Tables 5 and 6.

7.1.1 Piecewise Linear, Quadratic and Polynomial Models

In this section, we evaluate the performance of Ciper learning piecewise polynomial models for the classical (single-target) regression task. As already mentioned, we will first compare polynomial models and piecewise linear models, then polynomial models and piecewise quadratic models, and finally polynomial models and piecewise polynomial models.

MDLCiperX: Polynomial Models vs Piecewise Linear Models

MDLCiper learning general polynomial equations performs better than MDLCiperX generating piecewise linear equations (MDLCiperX with a limitation of the maximum non-binary degree to 1). In Figure 22, the blue circles are to the left of the red squares. Polynomial models perform better on 7 datasets, while piecewise linear equations performs better on 2 datasets. Given the outcome of the Wilcoxon signed-rank test, the difference in performance is statistically significant.

Table 20: A statistical comparison of the performance of polynomial models (learned by MDLCiper) and piecewise linear models (learned by MDLCiperX). Summary of Table 56.

MDLCiper : (piecewise linear)	*rrmse*	*ssc*	*mcl*
t-test	7:2	6:16	2:11
w-test	0.9665	0.0467	0.0002

MDLCiperX generating piecewise linear equations searches fewer equations than MDLCiper, where the difference in the number of equations considered is significant. This can be seen in Figure 22, where the red squares are placed lower as compared to the blue circles. The piecewise linear models (generated by MDLCiperX) are (statistically significantly) less complex than the polynomials learned by MDLCiper. This can be seen in Figure 22, where the red squares are smaller than the blue circles.

CVCiperX: Polynomial Models vs Piecewise Linear Models

The general polynomial equations learned by CVCiper and the piecewise linear equations learned by CVCiperX (CVCiperX with a limitation of the maximum non-binary degree to 1) have similar predictive performance. Polynomial models perform better on 9 datasets, while piecewise linear models perform better on 3 datasets. However, the outcome of the

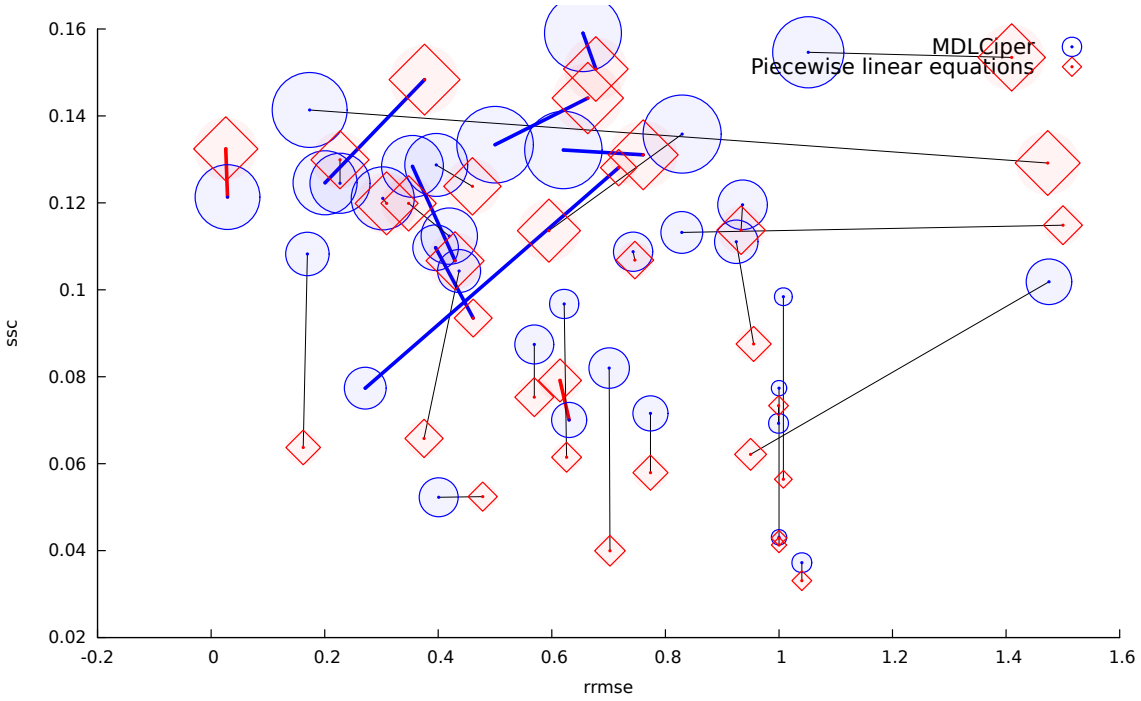


Figure 22: A visual comparison of the performance of polynomial models (MDLCIPER) and piecewise linear models (MDLCIPERX) on the single-target regression task.

Wilcoxon signed-rank test indicates that the difference in predictive performance is not statistically significant.

Table 21: A statistical comparison of the performance of polynomial models (learned by CVCIPER) and piecewise linear models (learned by CVCIPERX). Summary of Table 57.

CVCIPER : (piecewise linear)	*rrmse*	*ssc*	*mcl*
t-test	9:3	11:12	3:20
w-test	0.5491	0.7079	0.0

Less complex models (piecewise linear equations) are generated by CVCIPERX than by CVCIPER. This can be seen in Figure 23, where the red squares are smaller than the blue circles. The search spaces considered by the two approaches are of comparable size.

MDLCIPERX: Polynomial Models vs Piecewise Quadratic Models

MDLCIPERX generating piecewise quadratic equations and MDLCIPER have similar predictive performance. MDLCIPER performs better on 2 datasets. MDLCIPERX generating piecewise quadratic equations performs better on 2 datasets. Given the outcome of the Wilcoxon signed-rank test, the difference in performance is not statistically significant.

Table 22: A statistical comparison of the performance of polynomial models (MDLCIPER) and piecewise quadratic equations (MDLCIPERX). Summary of Table 58.

MDLCIPER : (piecewise quadratic)	*rrmse*	*ssc*	*mcl*
t-test	2:2	10:15	4:7
w-test	0.8482	0.1586	0.0181

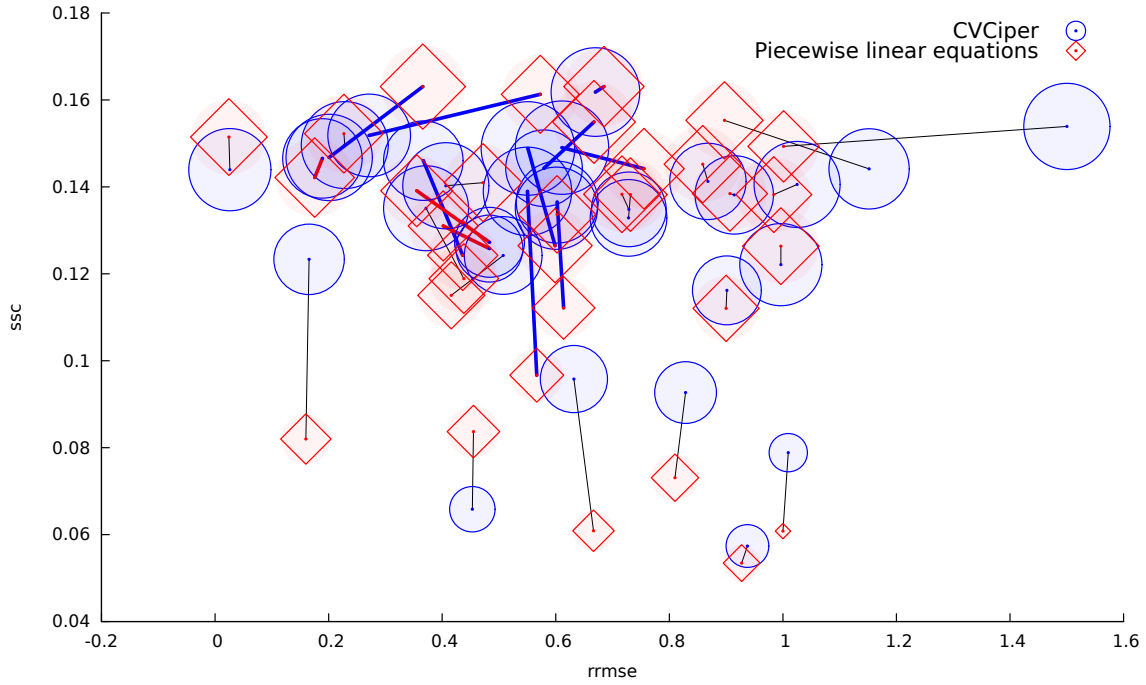


Figure 23: A visual comparison of the performance of polynomial models learned by CVCiper, and piecewise linear models on the single-target regression task.

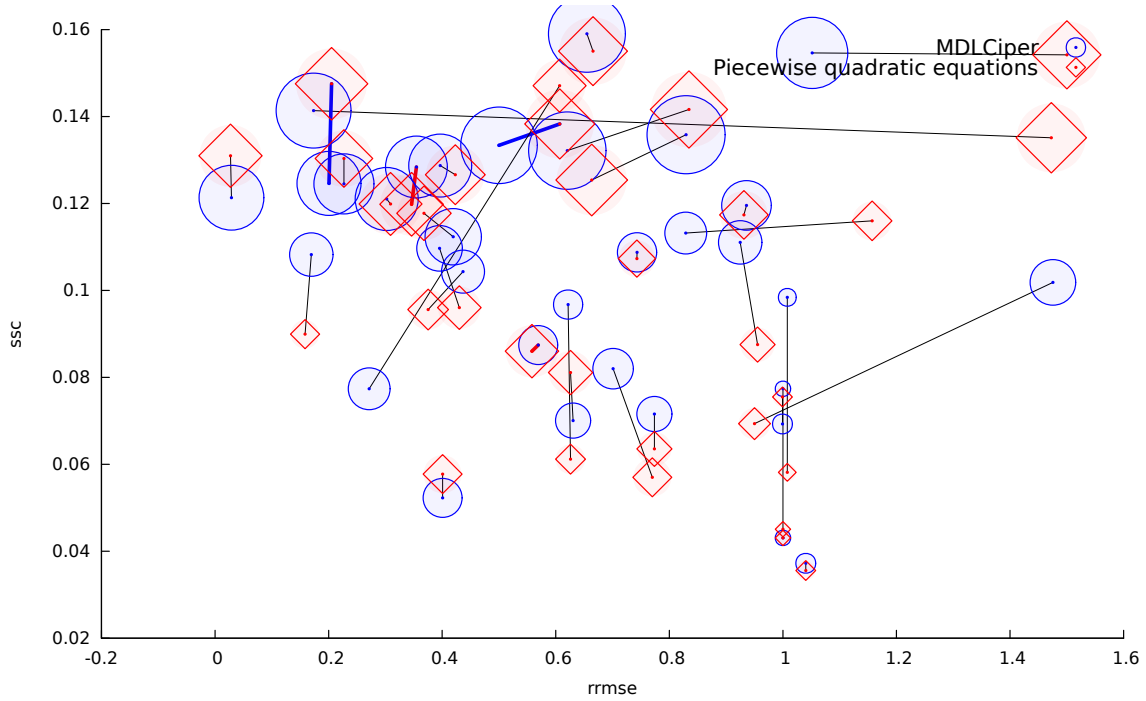


Figure 24: A visual comparison of the performance of polynomial models (MDLCiper) and piecewise quadratic equations (MDLCiperX) on the single-target regression task.

Less complex models are generated by MDLCiperX generating piecewise quadratic equations than by MDLCiper. This can be seen in Figure 24, where the red squares are smaller than the blue circles. The search space considered by the two approaches are of comparable size.

CVCIPERX: Polynomial Models vs Piecewise Quadratic Models

CVCIPERX that generates piecewise quadratic equations and CVCIPER have similar predictive performance. CVCIPER performs better on 5 datasets. CVCIPERX that generates piecewise quadratic equations performs better on 4 datasets. Given the outcome of the Wilcoxon signed-rank test, the difference in performance is not statistically significant.

Table 23: A statistical comparison of the performance of polynomial models (CVCIPER) and piecewise quadratic equations (CVCIPERX). Summary of Table 59.

CVCIPER : (piecewise quadratic)	*rrmse*	*ssc*	*mcl*
t-test	5:4	15:7	5:14
w-test	0.1209	0.9302	0.001

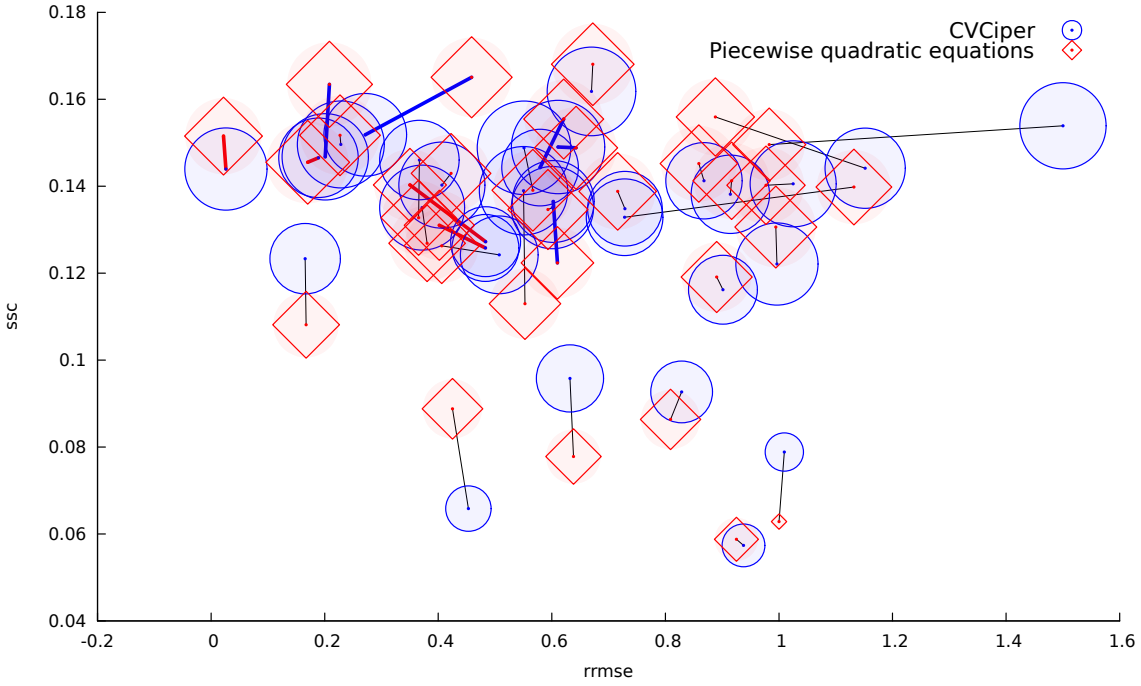


Figure 25: A visual comparison of the performance of polynomial models (CVCIPER) and piecewise quadratic models (CVCIPERX) on the single-target regression task.

The number of polynomial structures considered by CVCIPER is smaller than the number of piecewise models considered by CVCIPERX, but this difference is not significant. The polynomial models generated by CVCIPER are significantly more complex than the piecewise quadratic models learned by CVCIPERX. This can be seen in Figure 25, where the red squares are smaller than the blue circles.

MDLCIPERX: Polynomial Models vs Piecewise Polynomial Models

The piecewise polynomial models learned by MDLCIPERX and the polynomial models learned by MDLCIPER have similar predictive performance. The first perform better on 2 datasets and the second perform better on one dataset. The difference in performance is not statistically significant according to the Wilcoxon signed-rank test.

Table 24: A statistical comparison of the performance of polynomial models (learned by MDLCIPER) and piecewise polynomial models (learned by MDLCIPERX). Summary of Table 60.

MDLCIPER : MDLCIPERX	*rrmse*	*ssc*	*mcl*
t-test	2:1	15:5	1:7
w-test	0.7612	0.9548	0.0069

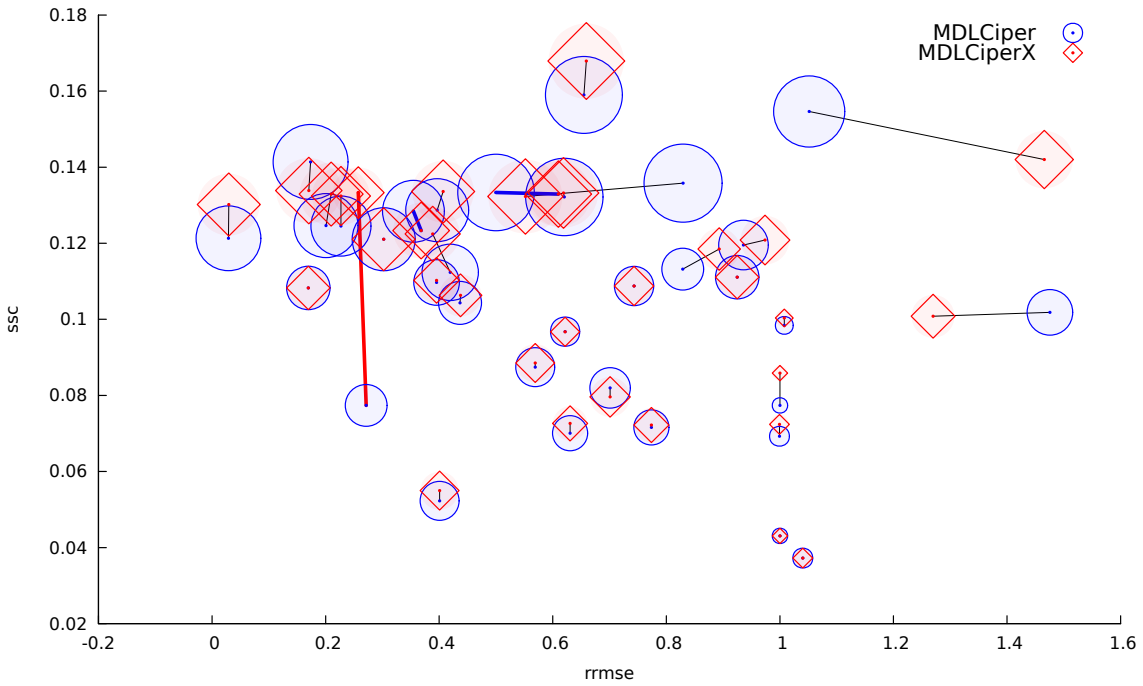


Figure 26: A visual comparison of the performance of polynomial models (MDLCIPER) and piecewise polynomial models (MDLCIPERX) on the single-target regression task.

MDLCIPER considers fewer polynomial models than MDLCIPERX piecewise polynomial models. This can be seen in Figure 26, where the blue circles are placed lower than the red squares. The piecewise polynomial models generated by MDLCIPERX are simpler than the polynomial models learned by MDLCIPER. This can be seen in Figure 26, where the red squares are smaller than the blue circles.

CVCIPERX: Polynomial Models vs Piecewise Polynomial Models

The piecewise polynomial models learned by CVCIPERX and the polynomial models learned by CVCIPER have similar predictive performance. The first perform better on one dataset and the second perform better on four datasets. The difference in performance is not statistically significant according to the Wilcoxon signed-rank test.

Table 25: A statistical comparison of the performance of polynomial models (learned by CVCIPER) and piecewise polynomial models (learned by CVCIPERX). Summary of Table 61.

CVCIPER : CVCIPERX	*rrmse*	*ssc*	*mcl*
t-test	1:4	15:2	3:7
w-test	0.4864	0.9988	0.0369

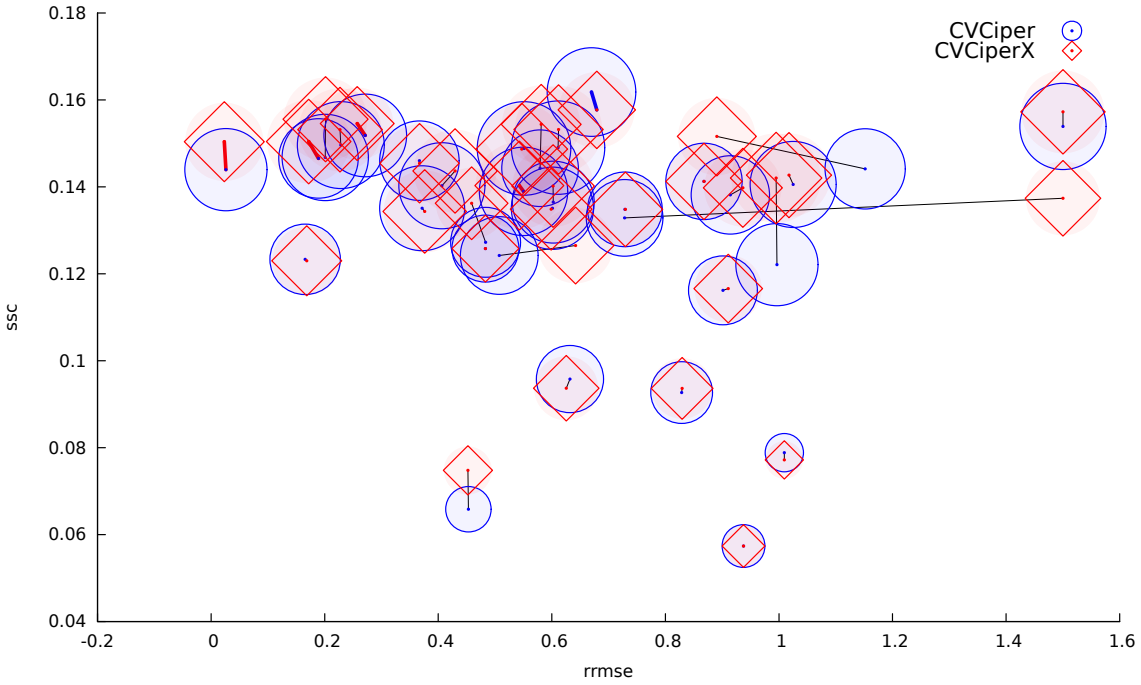


Figure 27: A visual comparison of the performance of polynomial models (CVCIPER) and piecewise polynomial models (CVCIPERX) on the single-target regression task.

CVCIPER considers fewer polynomial models than CVCIPERX piecewise polynomial models. This can be seen in Figure 27, where the blue circles are placed lower than the red squares. The piecewise polynomial models generated by CVCIPERX are simpler than the polynomial models learned by CVCIPER. This can be seen in Figure 27, where the red squares are smaller than the blue circles.

Summary: Piecewise Polynomial Models vs Polynomial Models

In general, linear piecewise models perform worse as compared to polynomial models. Quadratic piecewise models improve the predictive performance of linear piecewise models and have similar predictive performance as compared to polynomial equations. The quadratic and polynomial piecewise models are also less complex than polynomial models.

More models are explored when learning piecewise polynomial models as compared to learning polynomial models. When learning quadratic piecewise models, however, there is no statistically significant difference between the two approaches (CIPER and CIPERX), they explore similar numbers of models.

We have empirically proved that piecewise polynomial models with a lower degree (i.e., quadratic models) can have the same predictive performance as general polynomial models. In addition, similar numbers of models are explored during the learning of each type of model. Note that ordinary quadratic models perform worse than polynomial models (cf. Section 6.5) and that the piecewise extension is crucial for quadratic models to perform comparably to polynomial models.

7.1.2 CV vs MDL for learning piecewise polynomial models

In this section, we compare the performance of the two search heuristics within CIPER in the context of learning piecewise polynomial models. The maximum non-binary degree is not limited.

The piecewise polynomial models learned by CVCIPERX and MDLCIPERX have similar predictive performance. The ones learned by CVCIPERX perform better on 6 datasets, those

learned by MDLCIPERX perform better on 2 datasets. Given the outcome of the Wilcoxon signed-rank test, the difference in performance is not statistically significant.

Table 26: A statistical comparison of the performance of piecewise polynomial models learned by CVCIPERX and MDLCIPERX. Summary of Table 62.

CVCIPERX : MDLCIPERX	*rrmse*	*ssc*	*mcl*
t-test	6:2	1:30	0:31
w-test	0.7317	0.0	0.0

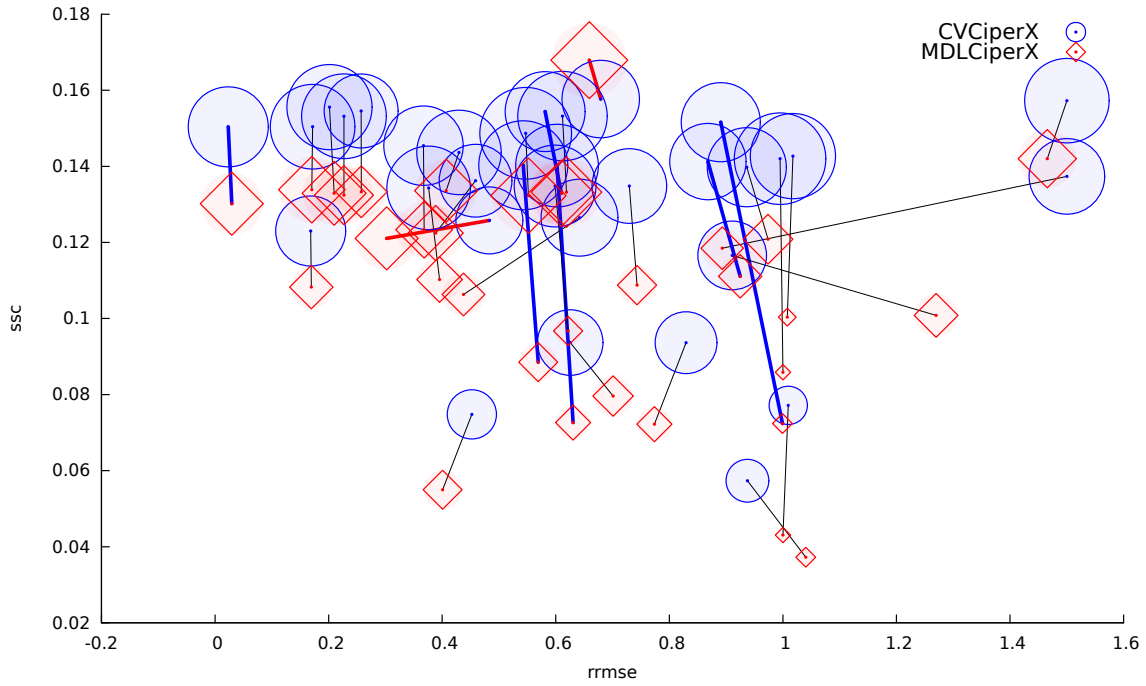


Figure 28: A visual comparison of the performance of piecewise polynomial models learned by CVCIPERX and MDLCIPERX on the single-target regression task.

MDLCIPERX searches a smaller space of models as compared to CVCIPERX. This can be seen in Figure 28, where the red squares are placed lower than the blue circles. Less complex models are generated by MDLCIPERX than by CVCIPERX. This can be seen in Figure 28, where the red squares are smaller than the blue circles.

We can conclude that MDLCIPERX and CVCIPERX have similar predictive performance. MDLCIPERX searches fewer models and produces simpler models. This conclusion mirrors the conclusion in Section 6.3 where we compared MDLCIPER and CVCIPER which learn polynomial models, on the single-target regression task.

7.1.3 Piecewise CIPER vs Other Regression Algorithms

In this section, we compare CVCIPERX and MDLCIPERX with other machine learning algorithms for regression. We use three regression algorithms implemented in the WEKA data mining software [24]. For the 10-fold cross validation, all datasets were split into folds using WEKA. In this way, the WEKA algorithms and our algorithm used exactly the same folds. The regression algorithms from the WEKA software that we used are: linear

regression (LR), regression trees[1] (RT), and model trees[2] (MT).

Table 27: A statistical comparison of the performance of MDLCIPERX with the performance of LR, RT, and MT. Summary of Table 63.

MDLCIPERX :	*LR*	*RT*	*MT*
t-test	13:1	13:1	2:4
w-test	0.9747	0.9956	0.071

Table 28: A statistical comparison of the performance of CVCIPERX with the performance of LR, RT, and MT. Summary of Table 64.

CVCIPERX :	*LR*	*RT*	*MT*
t-test	12:0	13:1	3:4
w-test	0.9971	0.9999	0.140

MDLCIPERX performs better than linear regression. MDLCIPER performs better on 13 datasets, linear regression performs better on 1 dataset. According to the Wilcoxon signed-rank test, the difference is statistically significant.

MDLCIPERX performs better than regression trees. MDLCIPERX performs better on 13 datasets and regression trees perform better on 1 dataset. According to the Wilcoxon signed-rank test, the difference is statistically significant.

MDLCIPERX and model trees have similar predictive performance. MDLCIPERX performs better on 2 datasets, model trees perform better on 4 datasets. According to the Wilcoxon signed-rank test, the difference is not statistically significant.

CVCIPERX performs better than linear regression. CVCIPERX performs better on 12 datasets. According to the Wilcoxon signed-rank test, the difference is statistically significant.

CVCIPERX performs better than regression trees. CVCIPERX performs better on 13 datasets, while regression trees perform better on 1 dataset. Given the outcome of the Wilcoxon signed-rank test, the difference is statistically significant.

CVCIPERX and model trees have similar predictive performance. CVCIPERX performs better on 3 datasets, while model trees perform better on 4 datasets. According to the Wilcoxon signed-rank test, the difference is not statistically significant.

We can conclude that MDLCIPERX and CVCIPERX generate models that have better predictive performance than linear regression. They also perform better than regression trees. Both MDLCIPERX and CVCIPERX generate models that have similar predictive performance with model trees.

7.2 Multi-target Regression

In this section, we compare MDLCIPER (CIPER with the improved MDL heuristic) and CVCIPER (CIPER with the CV heuristic) on the task of multi-target regression, described in Section 4.2.2. The attempt to predict multiple targets simultaneously should in theory reduce over-fitting. The use of different search heuristics should yield different results, i.e., different performance of the resulting models. For this comparison, we use the 10 multi-target regression datasets described in Table 7.

7.2.1 Polynomial models: CV vs MDL

We first compare polynomial models learned by CVCIPER and MDLCIPER for the multi-target regression task. CVCIPER performs better than MDLCIPER, with MDLCIPER being better on 2 datasets and CVCIPER performing better on 3 datasets. According to the Wilcoxon signed-rank test, the difference in performance is statistically significant. In Figure 29, the red squares are more to the left as compared to the blue circles.

[1]For a short introduction to regression trees, please see [4].
[2]For a short introduction to model trees, please see [65].

While the wins-losses difference is only one dataset, the difference is statistically significant. This can be explained as follows: For the multi-target case, each target is considered separately. CVCIPER wins on 3 datasets with a total of 36 targets, while MDLCIPER wins on two datasets with 5 targets.

Table 29: A statistical comparison of the performance of MDLCIPER and CVCIPER on the multi-target regression task. Summary of Table 65.

MDLCIPER : CVCIPER	*rrmse*	*ssc*	*mcl*
t-test	2:3	7:2	9:0
w-test	0.0	0.0536	1.0

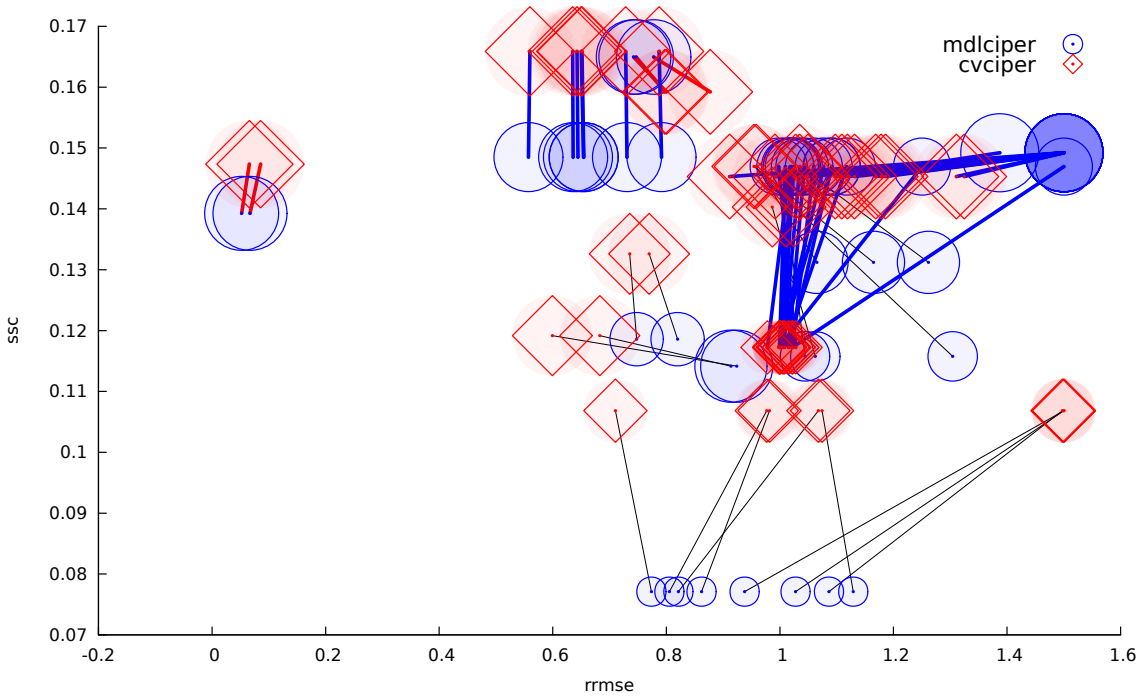


Figure 29: A visual comparison of the performance of the polynomial models learned by MDLCIPER and CVCIPER on the multi-target regression task.

MDLCIPER and CVCIPER have similar search space complexity. Less complex models are generated by MDLCIPER than by CVCIPER. This can be seen in Figure 29, where the blue circles are smaller than the red squares.

We can conclude that it is better to use CVCIPER than MDLCIPER for the multi-target regression task. We expect this to be true also for the classification task. We will consider the classification task in Section 7.3.

7.2.2 Piecewise models: CV vs MDL

We next compare piecewise polynomial models learned by CVCIPERX and MDLCIPERX.

CVCIPERX performs better than MDLCIPERX. MDLCIPERX performs better on 2 datasets (with a total of 10 targets), while CVCIPERX performs better on 1 dataset (with a total of 11 targets). According to the Wilcoxon signed-rank test, the difference is statistically significant. In Figure 30, the red circles are to the left of the blue circles. Less complex models are generated by MDLCIPERX. This can be seen in Figure 30, where the blue circles are smaller than the red circles.

Table 30: A statistical comparison of the performance of MDLCIPERX and CVCIPERX on the task of multi-target regression. Summary of Table 66.

MDLCIPERX : CVCIPERX	*rrmse*	*ssc*	*mcl*
t-test	2:1	7:1	9:0
w-test	0.0012	0.1487	1.0

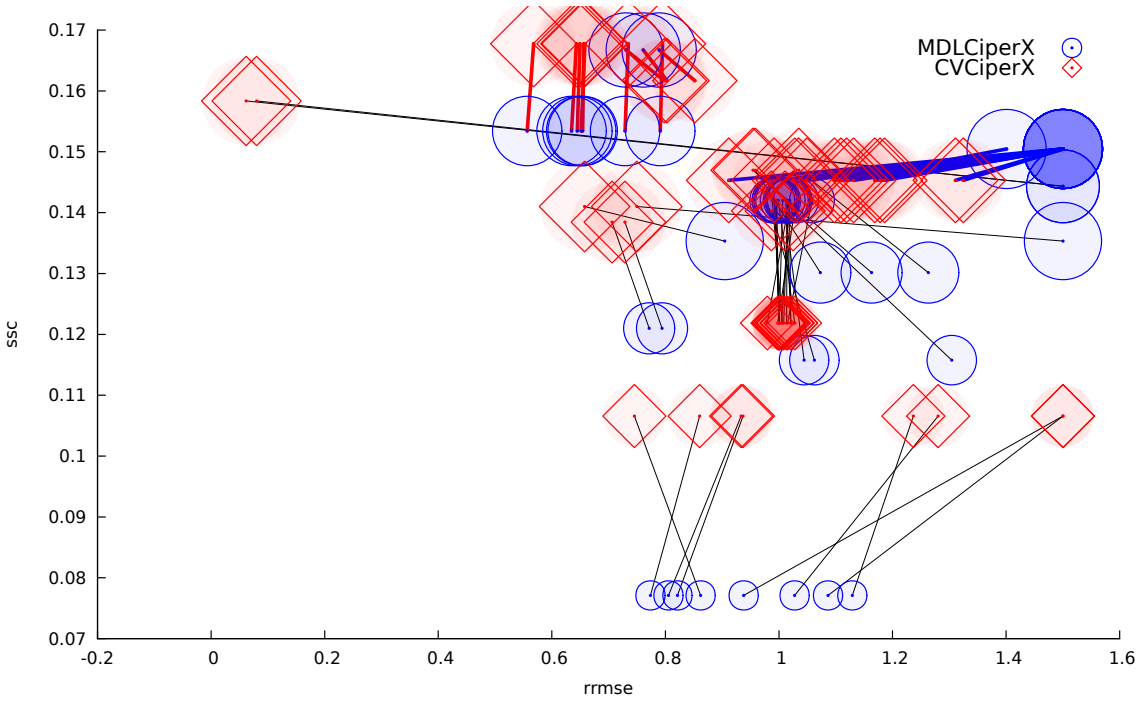


Figure 30: A visual comparison of the performance of the piecewise polynomial models learned by MDLCIPERX and CVCIPERX on the multi-target regression task.

We can conclude that CVCIPERX performs better than MDLCIPERX for the multi-target task. This conclusion is similar to the conclusion in Section 7.2, where we compared MDLCIPER and CVCIPER on the multi-target regression task.

7.3 Classification via Multi-Target Regression

In this section, we evaluate the performance of different versions of CIPER on the classification task. We will use the classification via regression approach to transform the task of classification into a multi-target regression task. After the multi-target regression task is solved with the multi-target version of CIPER, we use the generated model for classification via regression, as described in Section 4.2.3. The refinement operator used within all versions of CIPER is the new improved refinement operator described in Section 4.1.1. The heuristic functions used for all comparisons are the improved MDL heuristic described in Section 4.1.2 and the CV heuristic described in Section 4.1.3. For the comparisons, we use the 20 classification datasets described in Table 8. The performance on classification tasks is assessed in terms of classification error (*ce*, see Section 5.1.1.)

7.3.1 Evaluating the Effect of Degree

In this section, we evaluate the effect that the degree of the polynomial learned by the multi-target version of CIPER has on the performance of the algorithm. We first compare quadratic

equations (Ciper with a limitation of the maximum degree to 2) and linear equations (Ciper with a limitation of the maximum degree to 1) and then compare polynomial equations (Ciper with no limitation on the degree) and quadratic equations. This comparison is similar to the comparison performed in Section 6.5 and has the same goal, i.e., to empirically prove that using polynomial equations is better than using linear or quadratic equations.

MDLCiper: Quadratic vs Linear equations

When used in a classification via regression context, the algorithm that generates quadratic equations (MDLCiper with a limitation of the maximum degree to 2) performs better than the algorithm that generates linear equations (MDLCiper with a limitation of the maximum degree to 1). The algorithm that generates quadratic equations performs better on 7 datasets. According to the Wilcoxon signed-rank test, the difference is statistically significant. In Figure 31, the blue circles are to the left of the red squares.

Table 31: A statistical comparison of the performance of quadratic and linear equations learned by MDLCiper in a classification via multi-target regression context. Summary of Table 67.

quadratic : linear	*ce*	*ssc*	*mcl*
t-test	7:0	0:19	0:19
w-test	0.9843	0.0	0.0

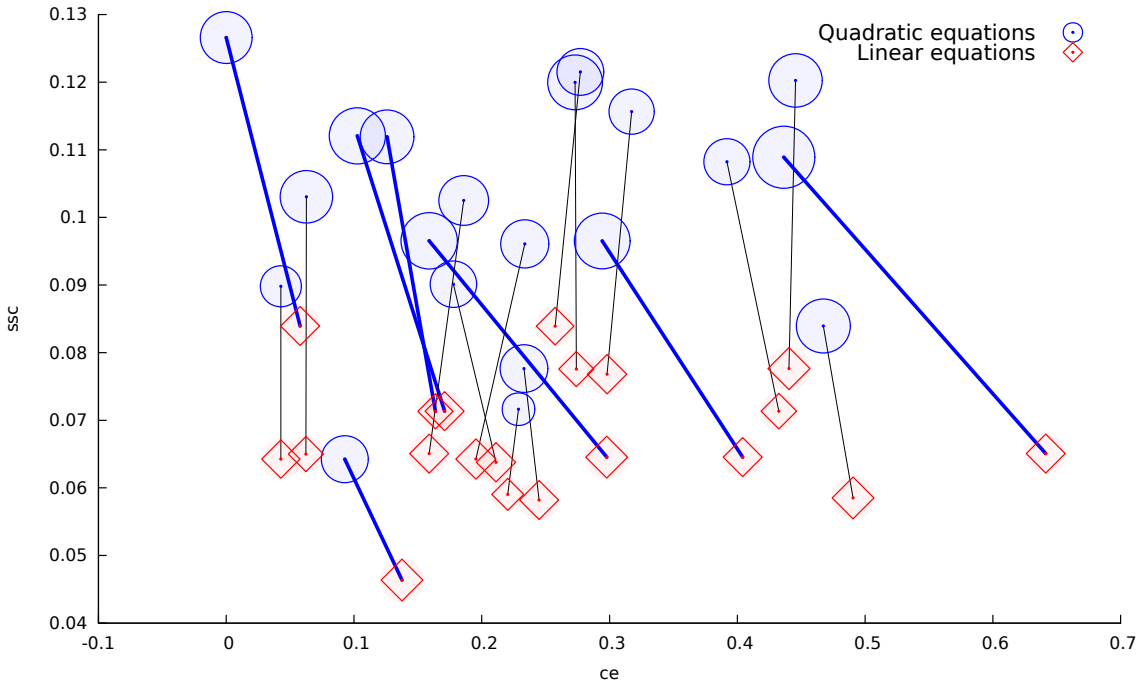


Figure 31: A visual comparison of the performance of quadratic and linear equations learned by MDLCiper in the context of the classification via multi-target regression task.

The algorithm that generates linear equations searches fewer equations than the algorithm that generates quadratic equations. This can also be seen in Figure 31, where the red squares are placed lower then the blue circles. Less complex models are generated by the algorithm that generates linear equations than by the algorithm that generates quadratic equations. This can be seen in Figure 31, where the red squares are smaller than the blue circles.

CVCIPER: Quadratic vs Linear equations

In the context via multi-target regression CVCIPER that generates quadratic equations performs better than CVCIPER that generates linear equations. CVCIPER that generates quadratic equations performs better on 8 datasets. According to the Wilcoxon signed-rank test, the difference is statistically significant. In Figure 32, the blue circles are to the left of the red squares.

Table 32: A statistical comparison of the performance of quadratic and linear equations learned by CVCIPER in the context of classification via multi-target regression. Summary of Table 68.

quadratic : linear	*ce*	*ssc*	*mcl*
t-test	8:0	0:20	0:20
w-test	0.9997	0.0	0.0

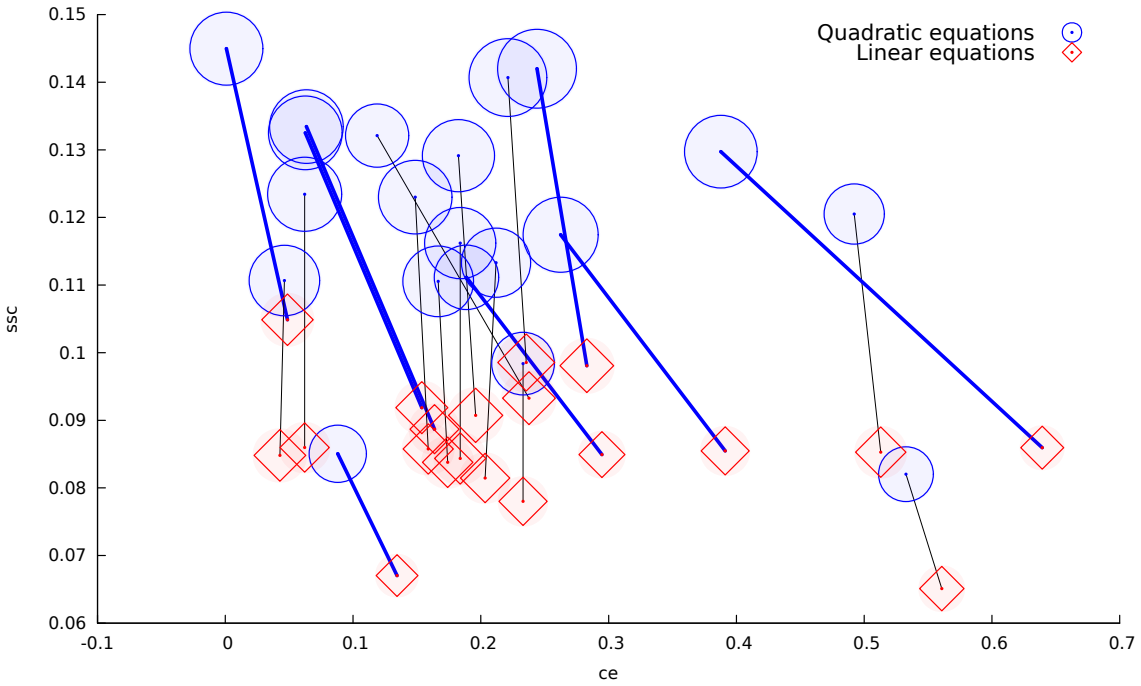


Figure 32: A visual comparison of the performance of quadratic and linear equations learned by CVCIPER in the context of the classification via multi-target regression task.

CVCIPER that generates linear equations searches fewer equations than CVCIPER that generates quadratic equations. This can be seen in Figure 32, where the red squares are placed lower then the blue circles. Less complex models are generated by CVCIPER that generates linear equations than by CVCIPER that generates quadratic equations. This can be seen in Figure 32, where the red squares are smaller than the blue circles.

MDLCIPER: Polynomial vs Quadratic Equations

MDLCIPER that generates polynomial equations (without limiting the degree) performs better than MDLCIPER that generates quadratic equations. Polynomial equations perform better on 3 datasets. According to the Wilcoxon signed-rank test, the difference is statistically significant. In Figure 33 the blue circles are left of the red squares.

MDLCIPER considers fewer quadratic equations then polynomial equations. This can be seen in Figure 33, where the red squares are placed lower the blue circles. The quadratic

Table 33: A statistical comparison of the performance of polynomial and quadratic equations learned by MDLCIPER in the context of classification via multi-target regression. Summary of Table 69.

polynomial : quadratic	*ce*	*ssc*	*mcl*
t-test	3:0	0:19	1:9
w-test	0.9723	0.0	0.0018

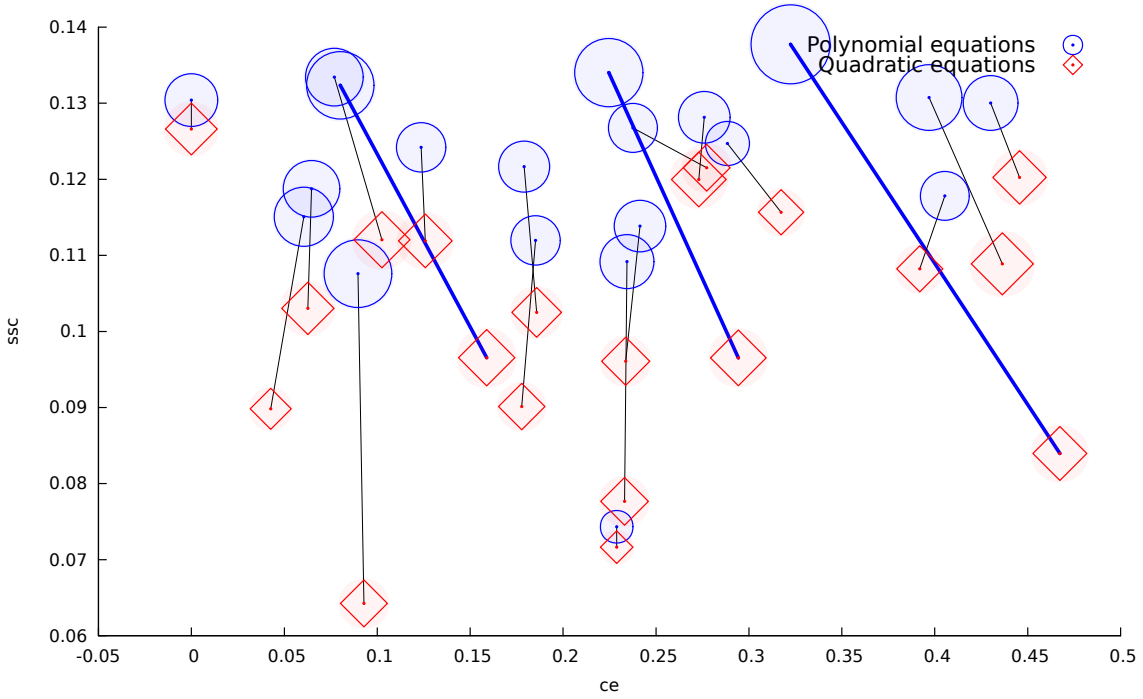


Figure 33: A visual comparison of the performance of polynomial and quadratic equations learned by MDLCIPER in the context of classification via multi-target regression.

equations are less complex than the polynomial equations. This can be seen in Figure 33, where the red squares are smaller than the blue circles.

CVCIPER: Polynomial vs Quadratic Equations

The polynomial equations generated by CVCIPER perform better than the corresponding quadratic equations. The polynomial equations perform better on 5 datasets. According to the Wilcoxon signed-rank test the difference is statistically significant. In Figure 34, the blue circles are to the left of the red squares.

Table 34: A statistical comparison of the performance of polynomial and quadratic equations learned by CVCIPER in the context of classification via multi-target regression. Summary of Table 70.

polynomial : quadratic	*ce*	*ssc*	*mcl*
t-test	5:0	0:20	0:19
w-test	0.9809	0.0	0.0

Fewer quadratic equations are searched by CVCIPER than polynomial equations. This can be seen in Figure 34, where the red squares are closer to the bottom as compared to the blue circles. The quadratic equations are less complex models. This can be seen in Figure

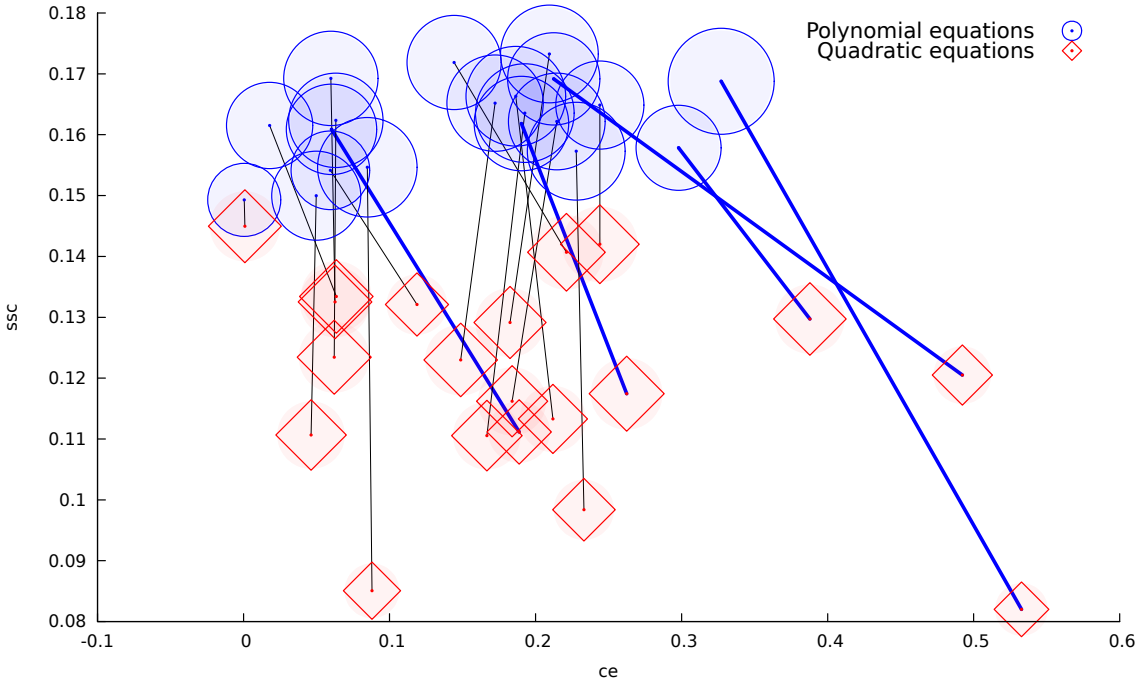


Figure 34: A visual comparison of the performance of polynomial and quadratic equations learned by CVCIPER in the context of classification via multi-target regression.

34, where the red squares are smaller than the blue circles.

Quadratic equations improve the predictive performance of CIPER as compared to linear equations. Polynomial equations improve the predictive performance of CIPER as compared to quadratic equations. More models are explored when learning quadratic equations than when learning linear equations. The resulting models are also more complex. We can conclude that it is better to use polynomial equations than quadratic or linear equations, and it is better to use quadratic equations than linear ones. Linear and quadratic equations are not always enough. In most cases, polynomials provide a reasonable and powerful extension.

7.3.2 CV vs MDL

In this section, we compare CVCIPER, i.e., CIPER using the CV heuristic proposed in Section 4.1.3, with MDLCIPER, i.e., CIPER using the MDL heuristic proposed in Section 4.1.2. Both are used to learn multi-target polynomial models. These are in turn used in the context of classification via regression.

CVCIPER performs better than MDLCIPER. CVCIPER performs better on 7 datasets. Given the outcome of the Wilcoxon signed-rank test, the difference in performance is statistically significant. In Figure 35, the blue circles are closer to the left then the red squares.

Table 35: A statistical comparison of the performance of CVCIPER and MDLCIPER learning polynomial models in the context of classification via multi-target regression. Summary of Table 71.

CVCIPER : MDLCIPER	*ce*	*ssc*	*mcl*
t-test	7:0	0:20	0:20
w-test	1.0	0.0	0.0

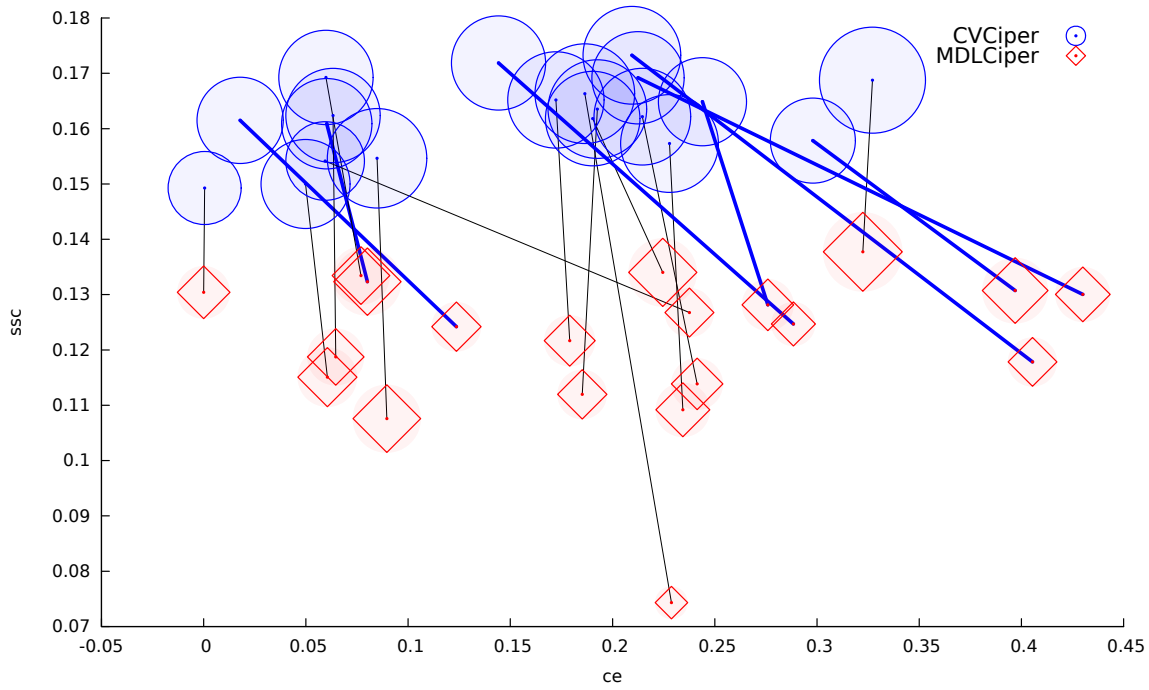


Figure 35: A visual comparison of the performance of CVCiper and MDLCiper learning polynomial models in the context of classification via multi-target regression.

MDLCiper searches fewer equations than CVCiper. This can be seen in Figure 35, where the red squares are closer to the bottom than the blue circles. Less complex models are generated by MDLCiper than by CVCiper. This can also be seen in Figure 35, where the red squares are smaller than the blue circles.

We can conclude that CVCiper performs better than MDLCiper for the classification via multi-target regression task. This was expected since CVCiper performs better than MDLCiper on the multi-target regression task. The small but significant difference in performance for multi-target regression is amplified in the classification via multi-target regression context.

7.3.3 Classification via Regression Algorithms

In this section, we compare classification via regression with Ciper to two other classification via regression algorithms, implemented in the WEKA data mining software [24]. All datasets were split into folds using WEKA. In this way, the WEKA algorithms and our algorithm used exactly the same folds.

WEKA implements a meta algorithm for classification via regression. This can be used with any regression algorithm in WEKA to perform the classification task. We compare classification via multi-target polynomial regression with CVCiper and MDLCiper to classification via linear regression (LR) and classification via model trees[3] (MT).

Classification via linear regression and MDLCiper have similar predictive performance. Classification via model trees (MT) performs better than MDLCiper. CVCiper performs better than classification via linear regression (LR). Classification via model trees (MT) and CVCiper have similar predictive performance.

[3]For more details please see [17].

Table 36: A statistical comparison of the performance of classification via regression with MDLCIPER, linear regression (LR), and model trees (MT). Summary of Table 72.

MDLCIPER :	*LR*	*MT*
t-test	5:2	1:4
w-test	0.6219	0.0181

Table 37: A statistical comparison of the performance of classification via regression with CVCIPER, linear regression (LR), and model trees (MT). Summary of Table 73.

CVCIPER :	*LR*	*MT*
t-test	7:0	1:1
w-test	0.9907	0.5719

7.3.4 Classification with CIPER vs Other Classification Algorithms

In this section, we compare CIPER to other classification algorithms as already described in Section 5.3. We compare classification via regression with CVCIPER and MDLCIPER with decision trees[4] (J48), Naive Bayes, and support vector machines[5] (SMO) with 3 different kernels: polynomial kernel with exponent 1 (SMO-E1), polynomial kernel with exponent 2 (SMO-E2) and a kernel with a radial basis function (SMO-KR).

Decision trees (J48) and classification via regression with MDLCIPER have similar predictive performance. Support vector machines with polynomial kernels with degree 1 (SMO-E1) and degree 2 (SMO-E2) perform better than MDLCIPER. MDLCIPER performs better than support vector machines with a radial basis function (SMO-KR). Naive Bayes and MDLCIPER have similar predictive performance.

Table 38: A statistical comparison of the performance of classification via regression with MDLCIPER to J48, SMO-E1, SMO-E2, SMO-KR, NaiveBayes. Summary of Table 74.

MDLCIPER :	*J48*	*SMO-E1*	*SMO-E2*	*SMO-KR*	*NaiveBayes*
t-test	2:5	0:3	0:6	6:1	5:1
w-test	0.318	0.028	0.0436	0.9525	0.7294

Classification via regression with CVCIPER performs better than decision trees (J48). Support vector machines with polynomial kernels with degree 1 (SMO-E1) and degree 2 (SMO-E2) have similar predictive performance to that of CVCIPER. CVCIPER performs better than support vector machines with a radial basis function (SMO-KR) and better than Naive Bayes.

Table 39: A statistical comparison of the performance of classification via regression with CVCIPER to J48, SMO-E1, SMO-E2, SMO-KR, NaiveBayes. Summary of Table 75.

CVCIPER :	*J48*	*SMO-E1*	*SMO-E2*	*SMO-KR*	*NaiveBayes*
t-test	4:1	1:0	1:1	11:0	11:0
w-test	0.9925	0.8988	0.6925	0.9984	0.9942

We can conclude that MDLCIPER performs better than some classification algorithms, i.e., support vector machines with a radial basis function (SMO-KR). It has similar predictive performance with decision trees (J48), Naive Bayes and classification via linear regression. It has worse predictive performance than support vector machines with polynomial kernels with degree 1 and 2 (SMO-E1, SMO-E2) and classification via model trees (MT).

[4]For introduction into decision trees, please see [45] and [44].
[5]For introduction into support vector machines, please see [9].

CVCIPER performs better than many classification algorithms: classification via linear regression (LR), one variant of support vector machines (SMO-KR), decision trees (J48), and Naive Bayes. CVCIPER has similar predictive performance with support vector machines with polynomial kernels with degree 1 and 2 (SMO-E1, SMO-E2), and classification via model trees (MT).

7.3.5 Classification via Regression with Piecewise Polynomial Models: CV vs MDL

In the context of classification via multi-target regression, CVCIPERX performs better than MDLCIPERX. CVCIPERX performs better on 7 datasets. Given the outcome of the Wilcoxon signed-rank test, the difference is statistically significant. In Figure 36, the blue circles are to the left of the red squares.

Table 40: A statistical comparison of the performance of CVCIPERX and MDLCIPERX learning piecewise polynomial models in the context of classification via multi-target regression. Summary of Table 76.

CVCIPERX : MDLCIPERX	*ce*	*ssc*	*mcl*
t-test	7:0	0:20	0:20
w-test	0.9996	0.0	0.0

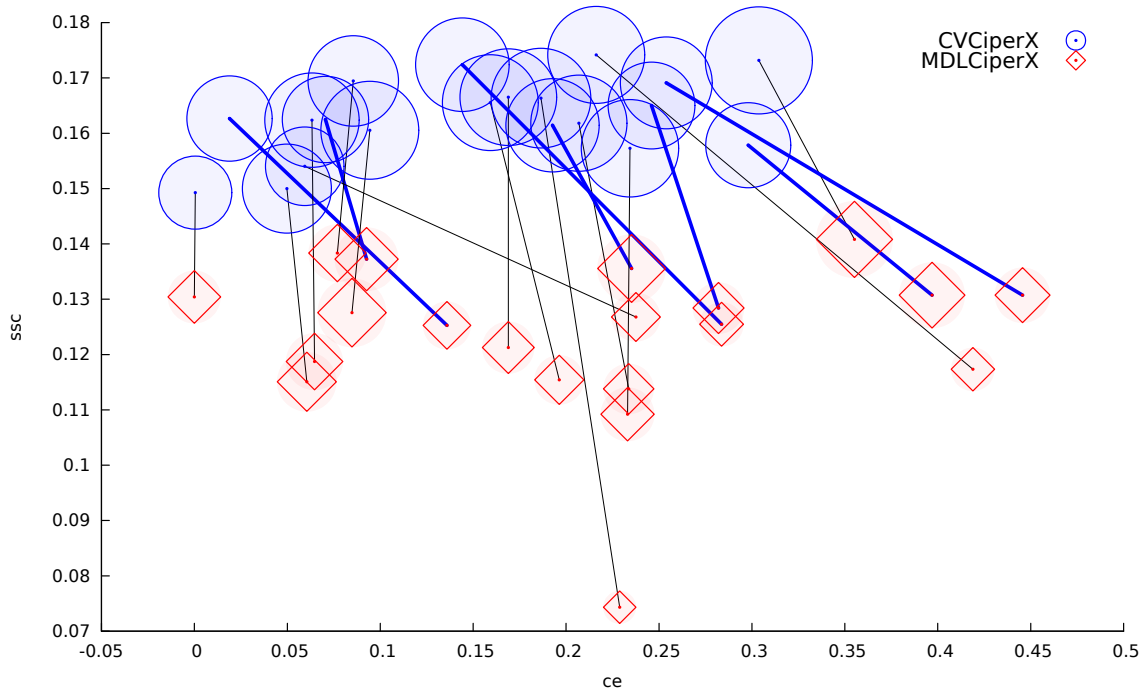


Figure 36: A visual comparison of the performance of CVCIPERX and MDLCIPERX learning piecewise polynomial models in the context of classification via multi-target regression.

MDLCIPERX searches fewer equations than CVCIPERX. This can be seen in Figure 36, where the red circles are closer to the bottom as compared to the blue circles. Less complex models are generated by MDLCIPERX than by CVCIPERX. This can be seen in Figure 36, where the red circles are smaller than the blue circles.

We can conclude that CVCIPERX performs better than MDLCIPERX. This was expected since CVCIPERX performs better than MDLCIPERX on the multi-target regression

task. The small but significant difference in performance for multi-target regression is amplified in the classification via multi-target regression context. The conclusion is similar to the conclusion of Section 7.3.2, where we compared MDLCiper and CVCiper on the classification task.

7.3.6 Piecewise Ciper vs Other Classification via Regression Algorithms

In this section, we compare classification via regression with CVCiperX and MDLCiperX to other classification via regression algorithms. WEKA implements a meta algorithm classification via regression, that can be used with any regression algorithms in WEKA to perform the classification task. We compare CVCiperX and MDLCiperX to classification via linear regression (LR) and classification via model trees[6] (MT).

Table 41: A statistical comparison of the performance of classification via regression with MDLCiperX, linear regression (LR), and model trees (MT). Summary of Table 77.

MDLCiperX :	*LR*	*MT*
t-test	5:3	1:4
w-test	0.5653	0.0133

Table 42: A statistical comparison of the performance of classification via regression with CVCiperX, linear regression(LR), and model trees (MT). Summary of Table 78.

CVCiperX :	*LR*	*MT*
t-test	6:0	1:2
w-test	0.994	0.444

Classification via linear regression (LR) and MDLCiperX have similar predictive performance. Classification via model trees (MT) performs better than MDLCiperX. CVCiperX performs better than classification via linear regression (LR). Classification via model trees (MT) and CVCiperX have similar predictive performance.

7.3.7 Piecewise Ciper vs Other Classification Algorithms

In this section, we compare classification via regression with CVCiperX and MDLCiperX to other classification algorithms. We compare CVCiperX and MDLCiperX with decision trees[7] (J48), Naive Bayes, and support vector machines[8] (SMO) with 3 different kernels: polynomial kernel with exponent 1 (SMO-E1), polynomial kernel with exponent 2 (SMO-E2) and a kernel with a radial basis function (SMO-KR).

Decision trees (J48) and MDLCiperX have similar predictive performance. Support vector machines with polynomial kernels with degree 1 (SMO-E1) and degree 2 (SMO-E2) perform better than MDLCiperX. MDLCiperX performs better than support vector machines with a radial basis functions SMO-KR. Naive Bayes and MDLCiperX have similar predictive performance.

Table 43: A statistical comparison of the performance of classification via regression with MDLCiperX and J48, SMO-E1, SMO-E2, SMO-KR, NaiveBayes. Summary of Table 79.

MDLCiperX :	*J48*	*SMO-E1*	*SMO-E2*	*SMO-KR*	*NaiveBayes*
t-test	2:4	0:3	0:6	6:1	5:1
w-test	0.2568	0.0183	0.0277	0.951	0.7294

CVCiperX performs better than decision trees (J48). Support vector machines with polynomial kernels with degree 1 (SMO-E1) and degree 2 (SMO-E2) have similar predictive

[6]For more details please see [17].

[7]For introduction into decision trees please see [45] and [44].

[8]For introduction into support vector machines please see [9].

performance to CVCIPERX. CVCIPERX performs better than support vector machines with a radial basis function (SMO-KR) as well as Naive Bayes.

Table 44: A statistical comparison of the performance of classification via regression with CVCIPERXand J48, SMO-E1, SMO-E2, SMO-KR, NaiveBayes. Summary of Table 75.

CVCIPERX :	*J48*	*SMO-E1*	*SMO-E2*	*SMO-KR*	*NaiveBayes*
t-test	4:2	1:0	1:1	11:0	10:0
w-test	0.9868	0.8988	0.6925	0.9984	0.9973

We can conclude that MDLCIPERX generates models that have better predictive performance than some classification algorithms, i.e., support vector machines with radial basis function (SMO-KR). It has similar predictive performance with decision trees (J48), Naive Bayes, and classification via linear regression. It has worse predictive performance than support vector machines with polynomial kernels with degree 1 and degree 2 (SMO-E1, SMO-E2) and classification via model trees (MT).

CVCIPERX generates models that have better predictive performance than many classification algorithms: classification via linear regression (LR), one variant of support vector machines (SMO-KR), decision trees (J48), and Naive Bayes. CVCIPERX generates models that have similar predictive performance with support vector machines with polynomial kernels with degree 1 and 2 (SMO-E1, SMO-E2), and classification via model trees (MT). The conclusions are consistent with the conclusions of Section 7.3.3 and Section 7.3.4 where we compared MDLCIPER and CVCIPER with the same classification algorithms.

7.4 Summary

This section summarizes the results from all previous sections of this chapter.

In Section 7.1 we have evaluated piecewise polynomial models. In Section 7.1.1 we have evaluated piecewise linear, quadratic and polynomial equations. We have empirically proved that piecewise polynomial models can generate models with lower degree than ordinary polynomial models with the same prediction performance as ordinary polynomial models.

Recall that we have already proved in Section 6.5 that quadratic equations perform worse than polynomial equations. To prove this we limited the degree of the non-binary variables. Then we compared polynomial models and piecewise linear models, then polynomial models and piecewise quadratic models, and in the end polynomial models and piecewise polynomial models.

Linear piecewise models perform worse as compared to polynomial models. Quadratic piecewise models improve the predictive performance and have performance similar to that of polynomial equations. Quadratic and polynomial piecewise models are also less complex than polynomial models. More models are explored when generating piecewise polynomial models than when learning polynomial models. But when generating quadratic piecewise models, there is no statistical difference between the two.

In Section 7.1.2 CVCIPERX (piecewise CVCIPER) and MDLCIPERX (piecewise MDLCIPER) were compared. On the regression task we concluded that MDLCIPERX and CVCIPERX have similar predictive performance. MDLCIPERX searched fewer models and produced simpler models.

In Section 7.1.3 we compared CVCIPERX and MDLCIPERX with other machine learning algorithms, implemented in the WEKA data mining software [24]: linear regression, regression trees, and model trees. We concluded that MDLCIPERX and CVCIPERX perform better than linear regression and regression trees. MDLCIPERX and CVCIPERX have

similar predictive performance with model trees.

In Section 7.2.1 MDLCiper (Ciper with the improved MDL heuristic) and CVCiper (Ciper with CV heuristic) were evaluated on the multi-target regression task. We concluded that CVCiper performs better than MDLCiper, but produces more complex models than MDLCiper. We concluded that it is better to use CVCiper than MDLCiper for the multi-target regression task. In Section 7.2.2, we evaluated piecewise Ciper and concluded that it is better to use CVCiperX than MDLCiperXfor the multi-target regression task.

Section 7.3 evaluates different variants of Ciper on classification tasks by applying the classification via multitarget regression approach. In Section 7.3.1 we evaluated the effect that the degree has on the performance of the algorithm for the classification task. This was done in two steps: we first compared quadratic and linear equations, then polynomial and quadratic equations. Quadratic equations improved the predictive performance of Ciper as compared to linear equations and polynomial equations improved the predictive performance of Ciper compared to quadratic equations. We concluded that it is better to use polynomial equations than quadratic and linear, and it is better to use quadratic equations than linear.

In Section 7.3.2, we compared CVCiper and MDLCiper, concluding that CVCiper performs better than MDLCiper on classification tasks. This was expected since CVCiper performs better than MDLCiper on the multi-target regression task.

In Section 7.3.3 and Section 7.3.4, we compared CVCiper and MDLCiper with other classification via regression and other classification algorithms. MDLCiper performs better than support vector machines with a radial basis function (SMO-KR). It has similar performance with decision trees (J48), Naive Bayes and classification via linear regression. It has worse predictive performance than support vector machines with polynomial kernels of degree 1 and 2 (SMO-E1 and SMO-E2) and classification via model trees.

CVCiper performs better than many classification algorithms: classification via linear regression, one variant of support vector machine with a radial basis function kernel (SMO-KR), decision trees (J48), and Naive Bayes. CVCiper has performance similar to support vector machines with polynomial kernels of degree 1 and 2 (SMO-E1, SMO-E2), and classification via model trees.

In Section 7.3.5, we evaluated piecewise Ciper on the classification task. We concluded that CVCiperX performs better than MDLCiperX. This is similar to the conclusion in Section 7.3.2, where we compared MDLCiper and CVCiper on the same task.

Finally, in Section 7.3.6 and Section 7.3.7 we compared CVCiperX and MDLCiperX with other classification via regression and other classification algorithms. MDLCiperX performs better than some classification algorithms. It has similar predictive performance with decision trees (J48), Naive Bayes and classification via linear regression. It has worse predictive performance than Support vector machine with polynomial kernels of degree 1 and 2 (SMO-E1, SMO-E2) and classification via model trees.

CVCiperX performs better than many classification algorithms: classification via linear regression, one variant of support vector machines (SMO-KR), decision trees (J48), and Naive Bayes. CVCiperX has predictive performance similar to that of support vector machines with polynomial kernels of degree 1 and 2 (SMO-E1, SMO-E2), and classification via model trees (MT).

8 Conclusions

8.1 Summary and discussion

In this thesis, we have addressed the task of polynomial regression, i.e., learning polynomial regression models from data. Polynomial models have been used extensively in the past, but they have been largely forgotten by the machine learning community. Recently, a machine learning algorithm CIPER for learning polynomial equations for regression has been developed and evaluated [61]. The algorithm has proved to be a good learner, being comparable to model trees and outperforming linear and stepwise regression. However, CIPER does have some limitations: a limited refinement operator, an ad-hoc heuristic function, no support for multiple targets, and no support for piecewise models (see Section 3.5). The main motivation for performing the work within this thesis was to overcome these limitations.

To this end, we have developed new methods that improve and extend the CIPER algorithm for polynomial regression. The improvements include new search space strategies (a refinement operator) and heuristic functions (MDL and CV based heuristics) for evaluating the performance of polynomial regression models. The extensions broaden the scope of polynomial regression toward piecewise and multi-target polynomial models and allow the use of polynomial models to perform classification via regression.

At this point, let us recall the central scientific hypotheses of this dissertation.

H1 The newly developed heuristics for evaluating polynomial regression models, based on a proper minimal description length scheme for polynomial regression and cross-validation estimates of a model's predictive performance on unseen data, improve the performance of the learned polynomial regression models.

H2 The new refinement operator for ordering the search space of candidate polynomials, that allows for larger changes of the candidate structure at each search step, improves the performance of the learned polynomial regression models.

H3 The polynomial regression models, induced with the approaches developed within the thesis, have predictive performance comparable to the performance of other commonly used regression algorithms.

H4 The classification models based on multi-target polynomial regression have predictive performance equal to or better than other classification via regression approaches as well as other classification models.

H5 Piecewise polynomial models of a limited degree achieve predictive performance comparable to the performance of polynomial regression models with unlimited degree.

To examine the validity of these hypotheses, we performed an extensive empirical evaluation and comparative analysis of the performance of the newly developed methods on a number of regression and classification tasks. Below, we summarize the main findings of the empirical evaluation and discuss how they confirm/support the hypotheses.

- The results presented in Section 6.1 show that the improved refinement operator that allows for larger changes of the current polynomial structure (introduced in Section 4.1.1) leads to more complex models that have better predictive performance than the models learned with the old refinement operator in the initial version of Ciper. This finding confirms the validity of the hypothesis *H2*. Note that the results also show that the improvement is related to the fact that Ciper with the improved refinement operator considers a larger number of candidate polynomials during the search.

- Section 6.2 presents the results of the evaluation of the improved MDL heuristic, introduced in Section 4.1.2 for evaluating candidate polynomial models. The results confirm hypothesis *H1* that the new MDL heuristic, developed in a principled manner, outperforms the old Ad-Hoc MDL heuristic. The models learned with the improved MDL heuristic are much simpler and have better predictive performance than the models learned with the initial version of Ciper.

- Taken together, the improved refinement operator and the new MDL heuristic lead to a smaller search space, simpler polynomial models, and better predictive performance.

- The results of the experiments presented in Section 6.3 show that the CV heuristic that uses cross-validation to evaluate the performance of polynomial models (introduced in Section 4.1.3) lead to models with performance comparable to the performance of the models learned with the improved MDL heuristics. This finding reconfirms the validity of the hypothesis *H1*. Note however, that in the context of multi-target regression, the models learned with the CV heuristic outperform the ones learned with the MDL heuristic (see the results presented in Section 7.2).

- The results presented in Section 6.5 and Section 7.1.1 show that general polynomials when used as regression models outperform linear and quadratic equations. This also holds in the context of classification via multi-target regression (Section 7.3.1).

- The results presented in Section 6.6 show that the newly developed version of Ciper perform slightly better or equally well as other commonly used regression algorithms. These results confirm the validity of the hypothesis *H3*.

- The results presented in Section 7.3 confirm the validity of the hypothesis *H4*. Multi-target polynomials when used as classification models perform better or equally well as classification models learned with other regression and classification algorithms.

- The results of piecewise polynomial models, presented in Section 7.1, show that they can achieve performance similar to the one of the general polynomial models at lower degrees. More specifically, linear piecewise models perform worse than the general ones, while quadratic piecewise models perform equally well as the general ones. At the same time, the piecewise polynomial models are simpler that the general polynomials. Note also that classification models based on piecewise polynomials perform better or equally well as classifiers learned with other classification methods. The results confirm the validity of the hypothesis *H5*.

In sum, all of our hypotheses are supported by the empirical evaluation and are thus confirmed by the investigations performed within this thesis.

8.2 Scientific contributions

We can now state the contributions of this dissertation to science, which closely match our initial expectations, presented in the introduction. The work presented in this thesis has lead to the following original contributions to the fields of machine learning and data mining:

- A new refinement operator that defines the search space of polynomial equations.
- New heuristic functions that guide the search through the space of polynomial equations. These include a heuristic based on an MDL encoding scheme for polynomial models and a heuristic based on cross-validated estimates of the predictive performance of polynomial model on unseen data.
- A generalization of simple polynomial regression models to more complex ones. This includes multi-target polynomial models for regression, piecewise polynomial models for single and multi-target regression, and the use of polynomial models for classification via regression.
- The design and implementation of a machine learning algorithm for inducing simple and complex regression polynomial models from data that incorporates all of the above elements.
- An extensive empirical evaluation of the newly developed algorithm (with all of its facets) and other existing algorithms on a large number of datasets for single and multi-target regression, as well as classification.

8.3 Further Work

As mentioned in Section 2.3, polynomial functions have certain limitations. One way of overcoming them is to consider the more general class of rational function models, i.e., ratios of two polynomial functions. They are a generalization of the polynomial models that includes the class of polynomial models as a subset (in the case when the denominator is a constant).

Rational function models have many advantages as compared to polynomial models. To mention a few, they can take an extremely wide range of shapes, they have better interpolation properties, and have excellent asymptotic properties. Rational functions with low degrees are commonly used as models of complicated structures and phenomena. However, they do have certain disadvantages as well: they are not understood as well as polynomials, and may have some unwanted asymptotes, but this should not be discouraging. Extending Ciper toward learning rational models is a promising venue for further work.

Another direction for future work is to consider alternative error metrics. In our current work, we focus our efforts on minimizing the sum of squared errors, i.e., squared differences between the observed and the predicted values of the dependent variable(s). Alternative error metrics consider absolute values of the differences. While this metric has already been used for linear regression, as in least absolute deviations (LAD), it has not been employed in the context of polynomial regression. LAD is known to be more robust with regard to outliers and may behave in the same way for polynomial regression.

9 Acknowledgements

I would like to show my gratitude to Dr. Jana Laganis, Peter Ljubič, Prof. Dr. Dončo Dimovski, my sister Sijče Pečkova, Dušana Majera, and Adnan Katirhan. I'm grateful to my supervisor Prof. Dr. Sašo Džeroski, my co-supervisor Prof. Dr. Ljupčo Todorovski, the Jožef Stefan International Postgraduate School, and to the Jožef Stefan Institute, for providing the environment in which I could work. I would like to thank my thesis committee, Prof. Dr. Bogdan Filipič, Prof. Dr. João Gama and Prof. Dr. Djani Juričić for their encouragement, comments, and questions. I am grateful to my parents for their constant support.

Several institutions provided funding for my research work. They include the Ad Futura Scientific and Educational Foundation, the Department of Knowledge Technologies (Jožef Stefan Institute, Ljubljana), and the European Commission (through the project IQ–Inductive Queries for Mining Patterns and Models).

I would like to thank everyone who helped me write this dissertation successfully.

10 References

[1] Akaike, H. A new look at the statistical model identification. *IEEE Transactions on Automatic Control* **19**, 716–723 (1974).

[2] Brazdil, P.; Gama, J.; Henery, B. Characterizing the applicability of classification algorithms using meta-level learning. In: *Proceedings of the 7th European Conference on Machine Learning*. 83–102 (Springer, Berlin, 1994).

[3] Breiman, L.; Spector, P. Submodel selection and evaluation in regression. *International Statistical Review* **60**, 291–319 (1992).

[4] Breiman, L.; Friedman, J. H.; Olshen, R. A.; Stone, C. J. *Classification and Regression Trees* (Wadsworth, Belmont, CA, 1984).

[5] Brien, C. J.; Payne, R. W. Tiers, structure formulae and the analysis of complicated experiments. *Journal of the Royal Statistical Society: Series D (The Statistician)* **48**, 41–52 (1999).

[6] Brown, G.; Wyatt, J.; Harris, R.; Yao, X. Diversity creation methods: a survey and categorisation. *Information Fusion* **6**, 5–20 (2005).

[7] Burnham, K. P.; Anderson, D. R. *Model Selection and Inference: A Practical Information-Theoretic Approach, 2nd edition* (Springer, New York, NY, 2002).

[8] Chatterjee, S.; Handcock, M. S.; Simonoff, J. S. *A Casebook for a First Course in Statistics and Data Analysis* (Wiley, New York, NY, 1995).

[9] Cristianini, N.; Shawe-Taylor, J. *An Introduction to Support Vector Machines (and other kernel-based learning methods)* (Cambridge University Press, Cambridge, 2000).

[10] Debeljak, M.; Kocev, D.; Towers, W.; Jones, M.; Griffiths, B.; Hallett, P. Potential of multi-objective models for risk-based mapping of the resilience characteristics of soils: demonstration at a national level. *Soil Use and Management* **25**, 66–77 (2009).

[11] Demšar, D.; Debeljak, M.; Lavigne, C.; Džeroski, S. Modelling pollen dispersal of genetically modified osr within the field. In: *Proceedings of the Annual Meeting of the Ecological Society of America*. 152 (Ecological Society of America, Washington, D.C., 2005).

[12] Diaconis, P.; Efron, B. Computer intensive methods in statistics. *Scientific American* **248**, 116–130 (1983).

[13] Džeroski, S.; Colbach, N.; Messean, A. Analysing the effect of field characteristics on gene flow between oilseed rape varieties and volunteers with regression trees. In: *Proceedings of the 2nd International Conference on Co-existence between GM and non-GM Based Agricultural Supply Chains*. 207–211 (Agropolis Productions, Montpellier, France, 2005).

[14] Džeroski, S.; Demšar, D.; Grbović, J. Predicting chemical parameters of river water quality from bioindicator data. *Applied Intelligence* **13**, 7–17 (2000).

[15] Evett, I. W.; Spiehler, E. J. *Rule induction in forensic science.* Technical report (Central Research Establishment, Home Office Forensic Science Service, Aldermaston, Reading, 1987).

[16] Frank, A.; Asuncion, A. UCI machine learning repository. `http://archive.ics.uci.edu/ml` (Accessed: July 2012).

[17] Frank, E.; Wang, Y.; Inglis, S.; Holmes, G.; Witten, I. H. Using model trees for classification. *Machine Learning* **32**, 63–76 (1998).

[18] Friedman, J. H. Multivariate adaptive regression splines. *The Annals of Statistics* **19**, 1–67 (1991).

[19] Friedman, M. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association* **32**, 675–701 (1937).

[20] Friedman, M. A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics* **11**, 86–92 (1940).

[21] Gashler, M. S.; Giraud-Carrier, C.; Martinez, T. Decision tree ensemble: Small heterogeneous is better than large homogeneous. In: *Proceedings of the Seventh International Conference on Machine Learning and Applications.* 900–905 (IEEE Computer Society, Washington, D.C., 2008).

[22] Ghahramani, Z. Unsupervised learning. In: *Advanced Lectures on Machine Learning.* 72–112 (Springer, New York, NY, 2004).

[23] Grünwald, P. D.; Myung, I. J.; Pitt, M. A. *Advances in Minimum Description Length: Theory and Applications* (MIT Press, Cambridge, MA, 2005).

[24] Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; Witten, I. H. The WEKA data mining software: an update. *SIGKDD Explorations Newsletter* **11**, 10–18 (2009).

[25] Hanley, J. A.; Shapiro, S. H. Sexual activity and the lifespan of male fruitflies: A dataset that gets attention. *Journal of Statistics Education* **2** (1994).

[26] Hansen, M. H.; Yu, B. Model selection and the principle of minimum description length. *Journal of the American Statistical Association* **96**, 746–774 (2001).

[27] Hastie, T.; Tibshirani, R.; Friedman, J. *The Elements of Statistical Learning: Data Mining, Inference and Prediction, 2nd edition* (Springer, New York, NY, 2009).

[28] Hill, T.; Lewicki, P. *STATISTICS: Methods and Applications* (StatSoft, Inc., Tulsa, OK, 2007).

[29] Hoerl, A. E.; Kennard, R. W. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics* **12**, 55–67 (1970).

[30] Hosmer, D. W.; Lemeshow, S. *Applied Logistic Regression, 2nd edition* (Wiley, New York, NY, 2000).

[31] Kalbfleisch, J. D.; Prentice, R. L. *The statistical Analysis of Failure Time Data* (Wiley, New York, NY, 1980).

[32] Karalič, A.; Bratko, I. First order regression. *Machine Learning* **26**, 147–176 (1997).

[33] Kibler, D.; Aha, D. W.; Albert, M. K. Instance-based prediction of real valued attributes. *Computational Intelligence* **5**, 51–57 (1989).

[34] King, R.; Muggleton, S.; Lewis, R.; Sternberg, M. Drug design by machine learning: the use of inductive logic programming to model the structure-activity relationships of trimethoprim analogues binding to dihydrofolate reductase. *Proceedings of the National Academy of Science* **89**, 11322–11326 (1992).

[35] King, R. D.; Hurst, J. D.; Sternberg, M. J. E. Comparison of artificial intelligence methods for modelling pharmaceutical QSARS. *Applied Artificial Intelligence* **9**, 213–233 (1995).

[36] Kocev, D.; Džeroski, S.; White, M. D.; Newell, G. R.; Griffioen, P. Using single- and multi-target regression trees and ensembles to model a compound index of vegetation condition. *Ecological Modelling* **220**, 1159–1168 (2009).

[37] Kocev, D.; Naumoski, A.; Mitreski, K.; Krstic, S.; Džeroski, S. Learning habitat models for the diatom community in Lake Prespa. *Ecological Modelling* **221**, 330–337 (2010).

[38] Kuncheva, L. I.; Whitaker, C. J. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning* **51**, 181–207 (2003).

[39] Langley, P. *Elements of Machine Learning* (Morgan Kaufmann, San Francisco, CA, 1995).

[40] McCullagh, P.; Nelder, J. A. *Generalized Linear Models, 2nd edition* (Chapman and Hall, London, 1989).

[41] Miller, A. J.; Shaw, D. E.; Veitch, L. G.; Smith, E. J. Analyzing the results of a cloud-seeding experiment in Tasmania. *Communications in Statistics - Theory and Methods* **A8**, 1017–1047 (1979).

[42] Neal, R. M. Assessing relevance determination methods using DELVE. In: Bishop, C. M (editor), *Neural Networks and Machine Learning*. 97–129 (Springer, New York, NY, 1998). `http://www.cs.toronto.edu/~delve/data/datasets.html` (Accessed: July 2012).

[43] Pace, R. K.; Barry, R. Sparse spatial autoregressions. *Statistics & Probability Letters* **33**, 291–297 (1997).

[44] Quinlan, J. *C4.5: Programs for Machine Learning* (Morgan Kaufmann, San Mateo, CA, 1993).

[45] Quinlan, J. R. Induction of decision trees. *Machine Learning* **1**, 81–106 (1986).

[46] Quinlan, J. R. Learning with continuous classes. In: *Proceedings of the 5th Australian Joint Conference on Artificial Intelligence*. 343–348 (World Scientific, Singapore, 1992).

[47] Rissanen, J. Fisher information and stochastic complexity. *IEEE Transactions on Information Theory* **42**, 40–47 (1996).

[48] Rissanen, J. MDL denoising. *IEEE Transactions on Information Theory* **46**, 2537–2543 (2000).

[49] Roos, T.; Myllymäki, P.; Tirri, H. On the behavior of MDL denoising. In: *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics*. 309–316 (Society for Artificial Intelligence and Statistics, Hastings, Barbados, 2005).

[50] Roos, T.; Myllymäki, P.; Rissanen, J. MDL denoising revisited. *IEEE Transactions on Signal Processing* **57**, 3347–3360 (2009).

[51] Runge, C.; Schlömilch, O. X.; Witzschel, B.; Cantor, M.; Kahl, E.; Mehmke, R. Über empirische Funktionen und die Interpolation zwischen äquidistanten Ordinaten. *Zeitschrift für Mathematik und Physik* **46**, 224–243 (1901).

[52] Schlimmer, J. C. *Concept Acquisition Through Representational Adjustment.* PhD thesis (University of California, Irvine, CA, 1987).

[53] Schwarz, G. Estimating the dimension of a model. *The Annals of Statistics* **6**, 461–464 (1978).

[54] Shtarkov, Y. M. Universal sequential coding of single messages. *Problems of Information Transmission* **23**, 175–186 (1987).

[55] Siegler, R. S. Three aspects of cognitive development. *Cognitive Psychology* **8**, 481–520 (1976).

[56] Simonoff, J. S. *Smoothing Methods in Statistics* (Springer, New York, NY, 1996).

[57] Smith, J. W.; Everhart, J. E.; Dickson, W. C.; Knowler, W. C.; Johannes, R. S. Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. In: *Proceedings of the Symposium on Computer Applications and Medical Care.* 261–265 (IEEE Computer Society, Washington, D.C., 1988).

[58] Sollich, P.; Krogh, A. Learning with ensembles: How overfitting can be useful. *Advances in Neural Information Processing Systems* **8**, 190–196 (1996).

[59] Stine, R. A. Model Selection Using Information Theory and the MDL Principle. *Sociological Methods Research* **33**, 230–260 (2004).

[60] Stone, M. H. The generalized Weierstrass approximation theorem. *Mathematics Magazine* **21**, 237–254 (1948).

[61] Todorovski, L.; Ljubič, P.; Džeroski, S. Inducing polynomial equations for regression. In: *Proceedings of the 15th European Conference of Machine Learning.* 441–452 (Springer, Berlin, 2004).

[62] Torgo, L. Regression data sets. `http://www.liaad.up.pt/~ltorgo/Regression/DataSets.html` (Accessed: July 2012).

[63] Turney, P. D. Robust classification with context-sensitive features. In: *Proceedings of the 6th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems.* 268–276 (Gordon & Breach Science Publishers, Edinburgh, 1993).

[64] Vlachos, P. StatLib datasets archive. `http://lib.stat.cmu.edu/` (Accessed: July 2012).

[65] Wang, Y.; Witten, I. H. Induction of model trees for predicting continuous classes. In: *Poster Papers of the 9th European Conference on Machine Learning.* 128–137 (Prague University of Economics, Prague, Czech Republic, 1997).

[66] Western, B. Vague theory and model uncertainty in macrosociology. *Sociological Methodology* **26**, 165–192 (1996).

[67] Wilcoxon, F. Individual comparisons by ranking methods. *Biometrics Bulletin* **1**, 80–83 (1945).

Index of Figures

Index of Tables

Appendices

A Evaluating CIPER Improvements: Complete Results

A.1 Evaluating the New Refinement Operator

Table 45: A comparison of the performance of CIPER using the new improved refinement operator (NewR) and old refinement operator (OldR). The performance is measured in terms of relative root mean squared error (*rrmse*), search space complexity (*ssc*), and model complexity (*mcl*) given with the equation length as explained in Chapter 5.

	rrmse			*ssc*			*mcl*		
dataset	NewR	t-test	OldR	NewR	t-test	OldR	NewR	t-test	OldR
2dplanes	0.228	$<$	0.262	$6.6 \cdot 10^6$	$>$	$2.5 \cdot 10^6$	$1.9 \cdot 10^3$	$>$	$1.6 \cdot 10^3$
auto-price	0.427	$=$	0.398	$2.8 \cdot 10^6$	$>$	$1.5 \cdot 10^6$	$9.5 \cdot 10^2$	$>$	$7.2 \cdot 10^2$
bank32nh	0.812	$<$	0.853	$7.7 \cdot 10^4$	$>$	$6.5 \cdot 10^3$	$3.0 \cdot 10^2$	$>$	$1.0 \cdot 10^2$
baskball	0.817	$=$	0.852	$9.5 \cdot 10^3$	$>$	$9.0 \cdot 10^2$	$3.0 \cdot 10^2$	$>$	$1.0 \cdot 10^2$
bodyfat	0.163	$=$	0.846	$2.8 \cdot 10^6$	$>$	$4.1 \cdot 10^5$	$1.3 \cdot 10^3$	$>$	$3.7 \cdot 10^2$
cal-housing	0.598	$=$	0.573	$2.7 \cdot 10^7$	$>$	$1.9 \cdot 10^7$	$6.0 \cdot 10^3$	$<$	$6.5 \cdot 10^3$
cpu-small	0.453	$=$	0.501	$2.6 \cdot 10^7$	$>$	$1.8 \cdot 10^7$	$4.4 \cdot 10^3$	$=$	$4.7 \cdot 10^3$
delta-ailerons	0.628	$<$	0.635	$2.1 \cdot 10^4$	$>$	$1.1 \cdot 10^3$	$3.0 \cdot 10^2$	$>$	$1.0 \cdot 10^2$
delta-elevators	0.757	$=$	0.758	$2.3 \cdot 10^4$	$>$	$1.3 \cdot 10^3$	$3.0 \cdot 10^2$	$>$	$1.0 \cdot 10^2$
elevators	0.676	$<$	0.766	$7.5 \cdot 10^4$	$>$	$3.7 \cdot 10^3$	$5.0 \cdot 10^2$	$>$	$2.0 \cdot 10^2$
elusage	0.401	$=$	0.401	$8.5 \cdot 10^4$	$>$	$2.3 \cdot 10^4$	$3.0 \cdot 10^2$	$=$	$3.0 \cdot 10^2$
fried-delve	0.200	$=$	0.200	$7.9 \cdot 10^6$	$>$	$6.1 \cdot 10^6$	$2.1 \cdot 10^3$	$<$	$3.1 \cdot 10^3$
house-8l	0.614	$<$	0.638	$2.7 \cdot 10^7$	$>$	$4.2 \cdot 10^6$	$8.0 \cdot 10^3$	$>$	$2.7 \cdot 10^3$
housing	0.407	$=$	0.425	$1.4 \cdot 10^7$	$>$	$8.4 \cdot 10^6$	$2.9 \cdot 10^3$	$=$	$2.9 \cdot 10^3$
kin8nm	0.846	$<$	0.852	$2.8 \cdot 10^4$	$>$	$1.7 \cdot 10^3$	$3.0 \cdot 10^2$	$>$	$1.0 \cdot 10^2$
mbagrade	0.938	$=$	0.956	$2.9 \cdot 10^3$	$>$	$5.0 \cdot 10^2$	$1.4 \cdot 10^2$	$>$	$1.0 \cdot 10^2$
mv	0.067	$<$	0.083	$1.7 \cdot 10^6$	$>$	$1.7 \cdot 10^6$	$1.4 \cdot 10^3$	$>$	$1.2 \cdot 10^3$
puma32h	0.884	$=$	0.884	$2.7 \cdot 10^4$	$>$	$6.5 \cdot 10^3$	$1.0 \cdot 10^2$	$=$	$1.0 \cdot 10^2$
pw-linear	0.333	$<$	0.455	$4.2 \cdot 10^6$	$>$	$1.1 \cdot 10^6$	$1.5 \cdot 10^3$	$>$	$1.1 \cdot 10^3$
pyrim	0.993	$=$	0.848	$6.6 \cdot 10^4$	$>$	$5.5 \cdot 10^3$	$2.9 \cdot 10^2$	$>$	$1.0 \cdot 10^2$
quake	1.000	$=$	1.000	$1.1 \cdot 10^3$	$>$	$7.0 \cdot 10^2$	$8.0 \cdot 10^1$	$=$	$8.0 \cdot 10^1$
triazines	0.984	$=$	1.011	$1.0 \cdot 10^5$	$>$	$1.2 \cdot 10^4$	$2.1 \cdot 10^2$	$>$	$1.0 \cdot 10^2$
vineyard	0.510	$=$	0.590	$2.4 \cdot 10^5$	$>$	$5.1 \cdot 10^4$	$4.0 \cdot 10^2$	$<$	$4.6 \cdot 10^2$

A.2 Evaluating the Improved MDL Heuristic

Table 46: The performance of CIPER using the improved refinement operator with the improved MDL heuristic (MDL) and the old ad-hoc heuristic (ad-hoc).

	rrmse			*ssc*			*mcl*		
dataset	MDL	t-test	ad-hoc	MDL	t-test	ad-hoc	MDL	t-test	ad-hoc
2dplanes	0.227	=	0.228	$8.9 \cdot 10^5$	<	$6.6 \cdot 10^6$	$2.3 \cdot 10^2$	<	$1.9 \cdot 10^3$
auto-price	0.396	=	0.427	$1.7 \cdot 10^5$	<	$2.8 \cdot 10^6$	$5.8 \cdot 10^1$	<	$9.5 \cdot 10^2$
bank32nh	0.655	<	0.812	$4.0 \cdot 10^7$	>	$7.7 \cdot 10^4$	$1.1 \cdot 10^3$	>	$3.0 \cdot 10^2$
baskball	0.774	=	0.817	$2.6 \cdot 10^3$	<	$9.5 \cdot 10^3$	$2.0 \cdot 10^1$	<	$3.0 \cdot 10^2$
bodyfat	0.170	=	0.163	$1.5 \cdot 10^5$	<	$2.8 \cdot 10^6$	$4.7 \cdot 10^1$	<	$1.3 \cdot 10^3$
cal-housing	0.829	=	0.598	$3.1 \cdot 10^6$	<	$2.7 \cdot 10^7$	$1.2 \cdot 10^3$	<	$6.0 \cdot 10^3$
cpu-small	0.174	<	0.453	$5.7 \cdot 10^6$	<	$2.6 \cdot 10^7$	$8.8 \cdot 10^2$	<	$4.4 \cdot 10^3$
delta-ailerons	0.569	<	0.628	$1.5 \cdot 10^4$	=	$2.1 \cdot 10^4$	$3.0 \cdot 10^1$	<	$3.0 \cdot 10^2$
delta-elevators	0.631	<	0.757	$2.2 \cdot 10^3$	<	$2.3 \cdot 10^4$	$2.0 \cdot 10^1$	<	$3.0 \cdot 10^2$
elevators	0.355	<	0.676	$1.4 \cdot 10^6$	>	$7.5 \cdot 10^4$	$2.7 \cdot 10^2$	<	$5.0 \cdot 10^2$
elusage	0.401	=	0.401	$3.1 \cdot 10^2$	<	$8.5 \cdot 10^4$	$3.0 \cdot 10^1$	<	$3.0 \cdot 10^2$
fried-delve	0.201	=	0.200	$9.0 \cdot 10^5$	<	$7.9 \cdot 10^6$	$3.3 \cdot 10^2$	<	$2.1 \cdot 10^3$
house-8l	0.620	=	0.614	$2.1 \cdot 10^6$	<	$2.7 \cdot 10^7$	$1.1 \cdot 10^3$	<	$8.0 \cdot 10^3$
housing	0.397	=	0.407	$1.4 \cdot 10^6$	<	$1.4 \cdot 10^7$	$2.9 \cdot 10^2$	<	$2.9 \cdot 10^3$
kin8nm	0.500	<	0.846	$2.4 \cdot 10^6$	>	$2.8 \cdot 10^4$	$1.0 \cdot 10^3$	>	$3.0 \cdot 10^2$
mbagrade	1.040	=	0.938	$5.6 \cdot 10^1$	<	$2.9 \cdot 10^3$	$2.0 \cdot 10^0$	<	$1.4 \cdot 10^2$
mv	0.029	<	0.067	$6.2 \cdot 10^5$	<	$1.7 \cdot 10^6$	$3.5 \cdot 10^2$	<	$1.4 \cdot 10^3$
puma32h	0.271	<	0.884	$5.0 \cdot 10^3$	<	$2.7 \cdot 10^4$	$4.0 \cdot 10^1$	<	$1.0 \cdot 10^2$
pw-linear	0.420	=	0.333	$2.3 \cdot 10^5$	<	$4.2 \cdot 10^6$	$1.6 \cdot 10^2$	<	$1.5 \cdot 10^3$
pyrim	0.829	=	0.993	$2.6 \cdot 10^5$	>	$6.6 \cdot 10^4$	$4.0 \cdot 10^1$	<	$2.9 \cdot 10^2$
quake	1.000	=	1.000	$5.0 \cdot 10^3$	>	$1.1 \cdot 10^3$	$0.0 \cdot 10^0$	<	$8.0 \cdot 10^1$
triazines	0.999	=	0.984	$2.0 \cdot 10^3$	<	$1.0 \cdot 10^5$	$2.0 \cdot 10^0$	<	$2.1 \cdot 10^2$
vineyard	0.701	=	0.510	$8.2 \cdot 10^3$	<	$2.4 \cdot 10^5$	$3.5 \cdot 10^1$	<	$4.0 \cdot 10^2$

A.3 Comparing the Two New Search Heuristics: CV vs MDL

Table 47: Comparison of the performance of CVCIPER - CIPER with the CV heuristic (CV) and MDLCIPER - CIPER with the MDL heuristic (MDL).

		rrmse			*ssc*			*mcl*	
dataset	CV	t-test	MDL	CV	t-test	MDL	CV	t-test	MDL
2dplanes	0.228	=	0.227	$1.4 \cdot 10^7$	>	$8.9 \cdot 10^5$	$2.6 \cdot 10^3$	>	$2.3 \cdot 10^2$
auto-price	0.372	=	0.396	$2.8 \cdot 10^6$	>	$1.7 \cdot 10^5$	$2.2 \cdot 10^3$	>	$5.8 \cdot 10^1$
bank32nh	0.670	>	0.655	$5.4 \cdot 10^7$	>	$4.0 \cdot 10^7$	$3.0 \cdot 10^3$	>	$1.1 \cdot 10^3$
baskball	0.829	=	0.774	$2.7 \cdot 10^4$	>	$2.6 \cdot 10^3$	$2.6 \cdot 10^2$	>	$2.0 \cdot 10^1$
bodyfat	0.166	=	0.170	$7.8 \cdot 10^5$	>	$1.5 \cdot 10^5$	$5.8 \cdot 10^2$	>	$4.7 \cdot 10^1$
cal-housing	0.551	=	0.829	$1.3 \cdot 10^7$	>	$3.1 \cdot 10^6$	$4.8 \cdot 10^3$	>	$1.2 \cdot 10^3$
cholesterol	1.025	=	1.008	$5.2 \cdot 10^6$	>	$5.0 \cdot 10^4$	$2.4 \cdot 10^3$	>	$1.0 \cdot 10^0$
cloud	0.507	=	0.437	$8.6 \cdot 10^5$	>	$9.6 \cdot 10^4$	$1.1 \cdot 10^3$	>	$4.5 \cdot 10^1$
cpu-small	0.189	>	0.174	$1.0 \cdot 10^7$	>	$5.7 \cdot 10^6$	$1.3 \cdot 10^3$	>	$8.8 \cdot 10^2$
delta-ailerons	0.550	<	0.569	$4.3 \cdot 10^6$	>	$1.5 \cdot 10^4$	$3.0 \cdot 10^3$	>	$3.0 \cdot 10^1$
delta-elevators	0.603	<	0.631	$3.3 \cdot 10^6$	>	$2.2 \cdot 10^3$	$1.7 \cdot 10^3$	>	$2.0 \cdot 10^1$
elevators	0.367	=	0.355	$9.5 \cdot 10^6$	>	$1.4 \cdot 10^6$	$1.3 \cdot 10^3$	>	$2.7 \cdot 10^2$
elusage	0.453	=	0.401	$1.4 \cdot 10^3$	>	$3.1 \cdot 10^2$	$5.7 \cdot 10^1$	>	$3.0 \cdot 10^1$
fried-delve	0.201	=	0.201	$1.0 \cdot 10^7$	>	$9.0 \cdot 10^5$	$2.5 \cdot 10^3$	>	$3.3 \cdot 10^2$
fruitfly	1.010	=	1.000	$5.8 \cdot 10^3$	=	$1.1 \cdot 10^2$	$2.8 \cdot 10^1$	=	$0.0 \cdot 10^0$
house-8l	0.611	=	0.620	$1.3 \cdot 10^7$	>	$2.1 \cdot 10^6$	$4.7 \cdot 10^3$	>	$1.1 \cdot 10^3$
housing	0.406	=	0.397	$5.0 \cdot 10^6$	>	$1.4 \cdot 10^6$	$2.4 \cdot 10^3$	>	$2.9 \cdot 10^2$
kin8nm	0.579	>	0.500	$7.8 \cdot 10^6$	>	$2.4 \cdot 10^6$	$1.0 \cdot 10^3$	=	$1.0 \cdot 10^3$
lowbwt	0.601	=	0.622	$2.8 \cdot 10^6$	>	$4.2 \cdot 10^4$	$1.7 \cdot 10^3$	>	$1.0 \cdot 10^1$
mbagrade	0.938	=	1.040	$5.5 \cdot 10^2$	>	$5.6 \cdot 10^1$	$4.4 \cdot 10^1$	>	$2.0 \cdot 10^0$
meta	1.500	=	1.052	$2.2 \cdot 10^7$	=	$2.4 \cdot 10^7$	$2.5 \cdot 10^3$	>	$6.2 \cdot 10^2$
mv	0.026	<	0.029	$7.5 \cdot 10^6$	>	$6.2 \cdot 10^5$	$1.7 \cdot 10^3$	>	$3.5 \cdot 10^2$
pharynx	0.729	=	0.743	$2.8 \cdot 10^6$	>	$1.6 \cdot 10^5$	$8.6 \cdot 10^2$	>	$3.1 \cdot 10^1$
puma32h	0.271	=	0.271	$1.8 \cdot 10^7$	>	$5.0 \cdot 10^3$	$1.8 \cdot 10^3$	>	$4.0 \cdot 10^1$
pw-linear	0.483	=	0.420	$1.2 \cdot 10^6$	>	$2.3 \cdot 10^5$	$5.6 \cdot 10^2$	>	$1.6 \cdot 10^2$
pyrim	0.728	=	0.829	$2.2 \cdot 10^6$	>	$2.6 \cdot 10^5$	$1.0 \cdot 10^3$	>	$4.0 \cdot 10^1$
quake	0.997	=	1.000	$6.8 \cdot 10^5$	>	$5.0 \cdot 10^3$	$1.7 \cdot 10^3$	>	$0.0 \cdot 10^0$
sensory	0.868	<	0.925	$5.6 \cdot 10^6$	>	$2.0 \cdot 10^5$	$10.0 \cdot 10^2$	>	$4.7 \cdot 10^1$
servo	0.483	>	0.302	$1.0 \cdot 10^6$	>	$6.1 \cdot 10^5$	$4.3 \cdot 10^2$	>	$3.0 \cdot 10^2$
strike	0.914	=	0.936	$4.0 \cdot 10^6$	>	$5.1 \cdot 10^5$	$1.2 \cdot 10^3$	>	$8.5 \cdot 10^1$
triazines	1.152	=	0.999	$7.7 \cdot 10^6$	>	$2.0 \cdot 10^3$	$1.4 \cdot 10^3$	>	$2.0 \cdot 10^0$
veteran	0.901	=	1.476	$3.5 \cdot 10^5$	>	$7.3 \cdot 10^4$	$5.1 \cdot 10^2$	>	$5.8 \cdot 10^1$
vineyard	0.632	=	0.701	$3.8 \cdot 10^4$	>	$8.2 \cdot 10^3$	$4.3 \cdot 10^2$	>	$3.5 \cdot 10^1$

A.4 Evaluating the Effect of Beam Size

Table 48: The performance of CIPER using the new refinement operator and the MDL heuristic with a large beam size (L) and a small beam size (S).

	rrmse			*ssc*			*mcl*		
dataset	L	t-test	S	L	t-test	S	L	t-test	S
2dplanes	0.227	=	0.227	$8.9 \cdot 10^5$	>	$8.0 \cdot 10^4$	$2.3 \cdot 10^2$	=	$2.3 \cdot 10^2$
auto-price	0.396	=	0.401	$1.7 \cdot 10^5$	>	$1.5 \cdot 10^4$	$5.8 \cdot 10^1$	>	$5.0 \cdot 10^1$
bank32nh	0.655	<	0.672	$4.0 \cdot 10^7$	>	$6.0 \cdot 10^5$	$1.1 \cdot 10^3$	>	$3.9 \cdot 10^2$
baskball	0.774	=	0.774	$2.6 \cdot 10^3$	=	$7.6 \cdot 10^2$	$2.0 \cdot 10^1$	=	$2.0 \cdot 10^1$
bodyfat	0.170	=	0.160	$1.5 \cdot 10^5$	>	$9.8 \cdot 10^3$	$4.7 \cdot 10^1$	=	$1.8 \cdot 10^1$
cal-housing	0.829	=	0.596	$3.1 \cdot 10^6$	>	$1.6 \cdot 10^5$	$1.2 \cdot 10^3$	>	$7.1 \cdot 10^2$
cholesterol	1.008	=	1.008	$5.0 \cdot 10^4$	>	$4.7 \cdot 10^2$	$1.0 \cdot 10^0$	=	$1.0 \cdot 10^0$
cloud	0.437	=	0.558	$9.6 \cdot 10^4$	>	$7.2 \cdot 10^3$	$4.5 \cdot 10^1$	=	$3.3 \cdot 10^1$
cpu-small	0.174	<	0.182	$5.7 \cdot 10^6$	>	$1.9 \cdot 10^5$	$8.8 \cdot 10^2$	>	$4.3 \cdot 10^2$
delta-ailerons	0.569	=	0.560	$1.5 \cdot 10^4$	=	$1.2 \cdot 10^4$	$3.0 \cdot 10^1$	<	$1.4 \cdot 10^2$
delta-elevators	0.631	>	0.608	$2.2 \cdot 10^3$	<	$1.0 \cdot 10^4$	$2.0 \cdot 10^1$	<	$1.1 \cdot 10^2$
elevators	0.355	>	0.341	$1.4 \cdot 10^6$	>	$4.1 \cdot 10^5$	$2.7 \cdot 10^2$	<	$4.0 \cdot 10^2$
elusage	0.401	=	0.401	$3.1 \cdot 10^2$	=	$3.1 \cdot 10^2$	$3.0 \cdot 10^1$	=	$3.0 \cdot 10^1$
fried-delve	0.201	=	0.201	$9.0 \cdot 10^5$	>	$1.2 \cdot 10^5$	$3.3 \cdot 10^2$	=	$3.2 \cdot 10^2$
fruitfly	1.000	=	1.000	$1.1 \cdot 10^2$	=	$1.1 \cdot 10^2$	$0.0 \cdot 10^0$	=	$0.0 \cdot 10^0$
house-8l	0.620	=	0.629	$2.1 \cdot 10^6$	>	$2.2 \cdot 10^5$	$1.1 \cdot 10^3$	>	$8.4 \cdot 10^2$
housing	0.397	=	0.442	$1.4 \cdot 10^6$	>	$1.0 \cdot 10^5$	$2.9 \cdot 10^2$	>	$2.2 \cdot 10^2$
kin8nm	0.500	<	0.604	$2.4 \cdot 10^6$	>	$8.7 \cdot 10^4$	$1.0 \cdot 10^3$	>	$4.0 \cdot 10^2$
lowbwt	0.622	=	0.626	$4.2 \cdot 10^4$	>	$8.8 \cdot 10^2$	$1.0 \cdot 10^1$	=	$1.1 \cdot 10^1$
mbagrade	1.040	=	1.040	$5.6 \cdot 10^1$	=	$4.6 \cdot 10^1$	$2.0 \cdot 10^0$	=	$2.0 \cdot 10^0$
meta	1.052	=	1.500	$2.4 \cdot 10^7$	>	$4.9 \cdot 10^5$	$6.2 \cdot 10^2$	>	$2.6 \cdot 10^2$
mv	0.029	<	0.032	$6.2 \cdot 10^5$	>	$8.0 \cdot 10^4$	$3.5 \cdot 10^2$	<	$4.1 \cdot 10^2$
pharynx	0.743	=	0.733	$1.6 \cdot 10^5$	>	$8.0 \cdot 10^3$	$3.1 \cdot 10^1$	>	$2.2 \cdot 10^1$
puma32h	0.271	=	0.271	$5.0 \cdot 10^3$	=	$5.0 \cdot 10^3$	$4.0 \cdot 10^1$	=	$4.0 \cdot 10^1$
pw-linear	0.420	<	0.590	$2.3 \cdot 10^5$	>	$4.2 \cdot 10^3$	$1.6 \cdot 10^2$	>	$4.7 \cdot 10^1$
pyrim	0.829	=	1.008	$2.6 \cdot 10^5$	>	$2.4 \cdot 10^4$	$4.0 \cdot 10^1$	=	$3.6 \cdot 10^1$
quake	1.000	=	1.000	$5.0 \cdot 10^3$	>	$7.0 \cdot 10^1$	$0.0 \cdot 10^0$	=	$0.0 \cdot 10^0$
sensory	0.925	=	0.955	$2.0 \cdot 10^5$	>	$5.4 \cdot 10^3$	$4.7 \cdot 10^1$	>	$2.0 \cdot 10^1$
servo	0.302	=	0.286	$6.1 \cdot 10^5$	>	$4.6 \cdot 10^4$	$3.0 \cdot 10^2$	>	$2.0 \cdot 10^2$
strike	0.936	=	0.966	$5.1 \cdot 10^5$	>	$5.2 \cdot 10^4$	$8.5 \cdot 10^1$	=	$7.7 \cdot 10^1$
triazines	0.999	=	0.999	$2.0 \cdot 10^3$	=	$1.7 \cdot 10^3$	$2.0 \cdot 10^0$	=	$2.0 \cdot 10^0$
veteran	1.476	=	0.941	$7.3 \cdot 10^4$	>	$2.3 \cdot 10^3$	$5.8 \cdot 10^1$	=	$1.5 \cdot 10^1$
vineyard	0.701	=	0.707	$8.2 \cdot 10^3$	>	$1.1 \cdot 10^3$	$3.5 \cdot 10^1$	=	$2.6 \cdot 10^1$

Table 49: The performance of CIPER using the new refinement operator and the CV heuristic with a large beam size (L) and a small beam size (S).

	rrmse			*ssc*			*mcl*		
dataset	L	t-test	S	L	t-test	S	L	t-test	S
2dplanes	0.228	$<$	0.252	$1.4 \cdot 10^7$	$>$	$9.5 \cdot 10^5$	$2.6 \cdot 10^3$	$>$	$1.5 \cdot 10^3$
auto-price	0.372	$=$	0.403	$2.8 \cdot 10^6$	$>$	$9.1 \cdot 10^4$	$2.2 \cdot 10^3$	$>$	$7.6 \cdot 10^2$
bank32nh	0.670	$<$	0.677	$5.4 \cdot 10^7$	$>$	$3.0 \cdot 10^6$	$3.0 \cdot 10^3$	$>$	$1.5 \cdot 10^3$
baskball	0.829	$=$	0.829	$2.7 \cdot 10^4$	$>$	$3.3 \cdot 10^3$	$2.6 \cdot 10^2$	$>$	$1.5 \cdot 10^2$
bodyfat	0.166	$>$	0.160	$7.8 \cdot 10^5$	$>$	$2.9 \cdot 10^4$	$5.8 \cdot 10^2$	$>$	$1.4 \cdot 10^2$
cal-housing	0.551	$<$	0.585	$1.3 \cdot 10^7$	$>$	$6.2 \cdot 10^5$	$4.8 \cdot 10^3$	$>$	$2.3 \cdot 10^3$
cholesterol	1.025	$=$	0.996	$5.2 \cdot 10^6$	$>$	$1.4 \cdot 10^5$	$2.4 \cdot 10^3$	$>$	$7.1 \cdot 10^2$
cloud	0.507	$=$	0.502	$8.6 \cdot 10^5$	$>$	$4.1 \cdot 10^4$	$1.1 \cdot 10^3$	$>$	$4.4 \cdot 10^2$
cpu-small	0.189	$<$	0.256	$1.0 \cdot 10^7$	$>$	$4.5 \cdot 10^5$	$1.3 \cdot 10^3$	$>$	$7.9 \cdot 10^2$
delta-ailerons	0.550	$=$	0.555	$4.3 \cdot 10^6$	$>$	$1.7 \cdot 10^5$	$3.0 \cdot 10^3$	$>$	$1.2 \cdot 10^3$
delta-elevators	0.603	$<$	0.611	$3.3 \cdot 10^6$	$>$	$2.1 \cdot 10^5$	$1.7 \cdot 10^3$	$>$	$1.0 \cdot 10^3$
elevators	0.367	$<$	0.430	$9.5 \cdot 10^6$	$>$	$5.1 \cdot 10^5$	$1.3 \cdot 10^3$	$>$	$8.0 \cdot 10^2$
elusage	0.453	$=$	0.453	$1.4 \cdot 10^3$	$=$	$1.4 \cdot 10^3$	$5.7 \cdot 10^1$	$=$	$5.7 \cdot 10^1$
fried-delve	0.201	$<$	0.230	$1.0 \cdot 10^7$	$>$	$8.0 \cdot 10^5$	$2.5 \cdot 10^3$	$>$	$1.9 \cdot 10^3$
fruitfly	1.010	$=$	1.009	$5.8 \cdot 10^3$	$=$	$1.4 \cdot 10^3$	$2.8 \cdot 10^1$	$=$	$2.2 \cdot 10^1$
house-8l	0.611	$<$	0.638	$1.3 \cdot 10^7$	$>$	$5.7 \cdot 10^5$	$4.7 \cdot 10^3$	$>$	$2.3 \cdot 10^3$
housing	0.406	$<$	0.472	$5.0 \cdot 10^6$	$>$	$1.9 \cdot 10^5$	$2.4 \cdot 10^3$	$>$	$1.0 \cdot 10^3$
kin8nm	0.579	$<$	0.613	$7.8 \cdot 10^6$	$>$	$5.2 \cdot 10^5$	$1.0 \cdot 10^3$	$>$	$6.9 \cdot 10^2$
lowbwt	0.601	$=$	0.611	$2.8 \cdot 10^6$	$>$	$7.2 \cdot 10^4$	$1.7 \cdot 10^3$	$>$	$4.8 \cdot 10^2$
mbagrade	0.938	$=$	0.938	$5.5 \cdot 10^2$	$=$	$5.3 \cdot 10^2$	$4.4 \cdot 10^1$	$=$	$4.4 \cdot 10^1$
meta	1.500	$=$	0.973	$2.2 \cdot 10^7$	$>$	$3.9 \cdot 10^5$	$2.5 \cdot 10^3$	$>$	$5.6 \cdot 10^2$
mv	0.026	$<$	0.060	$7.5 \cdot 10^6$	$>$	$5.6 \cdot 10^5$	$1.7 \cdot 10^3$	$>$	$1.0 \cdot 10^3$
pharynx	0.729	$=$	0.729	$2.8 \cdot 10^6$	$>$	$8.0 \cdot 10^4$	$8.6 \cdot 10^2$	$>$	$2.9 \cdot 10^2$
puma32h	0.271	$=$	0.275	$1.8 \cdot 10^7$	$>$	$9.2 \cdot 10^5$	$1.8 \cdot 10^3$	$>$	$8.4 \cdot 10^2$
pw-linear	0.483	$<$	0.584	$1.2 \cdot 10^6$	$>$	$4.7 \cdot 10^4$	$5.6 \cdot 10^2$	$>$	$3.2 \cdot 10^2$
pyrim	0.728	$=$	1.162	$2.2 \cdot 10^6$	$>$	$8.8 \cdot 10^4$	$1.0 \cdot 10^3$	$>$	$4.4 \cdot 10^2$
quake	0.997	$=$	0.996	$6.8 \cdot 10^5$	$>$	$1.1 \cdot 10^5$	$1.7 \cdot 10^3$	$>$	$1.5 \cdot 10^3$
sensory	0.868	$<$	0.905	$5.6 \cdot 10^6$	$>$	$2.2 \cdot 10^5$	$10.0 \cdot 10^2$	$>$	$5.3 \cdot 10^2$
servo	0.483	$=$	0.556	$1.0 \cdot 10^6$	$>$	$3.5 \cdot 10^4$	$4.3 \cdot 10^2$	$>$	$2.3 \cdot 10^2$
strike	0.914	$=$	0.921	$4.0 \cdot 10^6$	$>$	$2.1 \cdot 10^5$	$1.2 \cdot 10^3$	$>$	$6.4 \cdot 10^2$
triazines	1.152	$=$	1.153	$7.7 \cdot 10^6$	$>$	$2.0 \cdot 10^5$	$1.4 \cdot 10^3$	$>$	$4.7 \cdot 10^2$
veteran	0.901	$=$	0.915	$3.5 \cdot 10^5$	$>$	$1.4 \cdot 10^4$	$5.1 \cdot 10^2$	$>$	$2.2 \cdot 10^2$
vineyard	0.632	$=$	0.655	$3.8 \cdot 10^4$	$>$	$4.1 \cdot 10^3$	$4.3 \cdot 10^2$	$>$	$2.0 \cdot 10^2$

A.5 Evaluating the Effect of Degree

Table 50: The performance of CIPER is using the new refinement operator and the improved MDL heuristic when learning quadratic equations - with a maximum degree of 2 (Q) and linear equations - with a maximum degree of 1 (L).

	rrmse			*ssc*			*mcl*		
dataset	Q	t-test	L	Q	t-test	L	Q	t-test	L
2dplanes	0.562	$<$	0.621	$1.0 \cdot 10^4$	$>$	$6.2 \cdot 10^2$	$1.1 \cdot 10^2$	$>$	$3.0 \cdot 10^1$
auto-price	0.458	$=$	0.474	$3.0 \cdot 10^4$	$>$	$1.2 \cdot 10^3$	$4.1 \cdot 10^1$	$>$	$2.3 \cdot 10^1$
bank32nh	0.671	$<$	0.772	$3.2 \cdot 10^5$	$>$	$2.6 \cdot 10^3$	$3.3 \cdot 10^2$	$>$	$4.0 \cdot 10^1$
baskball	0.774	$=$	0.774	$5.4 \cdot 10^2$	$>$	$1.0 \cdot 10^2$	$2.0 \cdot 10^1$	$=$	$2.0 \cdot 10^1$
bodyfat	0.160	$=$	0.162	$2.0 \cdot 10^4$	$>$	$1.1 \cdot 10^3$	$1.2 \cdot 10^1$	$<$	$1.9 \cdot 10^1$
cal-housing	0.648	$=$	0.647	$5.1 \cdot 10^3$	$=$	$6.0 \cdot 10^2$	$5.8 \cdot 10^1$	$>$	$3.0 \cdot 10^1$
cholesterol	1.008	$=$	1.008	$2.5 \cdot 10^4$	$>$	$1.3 \cdot 10^3$	$1.0 \cdot 10^0$	$=$	$1.0 \cdot 10^0$
cloud	0.383	$=$	0.383	$9.2 \cdot 10^3$	$>$	$3.7 \cdot 10^2$	$2.9 \cdot 10^1$	$=$	$2.9 \cdot 10^1$
cpu-small	0.221	$<$	0.549	$2.0 \cdot 10^4$	$>$	$1.1 \cdot 10^3$	$1.3 \cdot 10^2$	$>$	$4.0 \cdot 10^1$
delta-ailerons	1.000	$=$	1.000	$1.1 \cdot 10^2$	$=$	$6.0 \cdot 10^1$	$0.0 \cdot 10^0$	$=$	$0.0 \cdot 10^0$
delta-elevators	1.000	$=$	1.000	$1.3 \cdot 10^2$	$=$	$7.0 \cdot 10^1$	$0.0 \cdot 10^0$	$=$	$0.0 \cdot 10^0$
elevators	1.000	$=$	1.000	$3.3 \cdot 10^2$	$=$	$1.9 \cdot 10^2$	$0.0 \cdot 10^0$	$=$	$0.0 \cdot 10^0$
elusage	0.401	$=$	0.478	$1.4 \cdot 10^2$	$=$	$4.0 \cdot 10^1$	$3.0 \cdot 10^1$	$=$	$1.0 \cdot 10^1$
fried-delve	0.566	$<$	0.599	$1.0 \cdot 10^4$	$>$	$6.2 \cdot 10^2$	$1.1 \cdot 10^2$	$>$	$3.0 \cdot 10^1$
fruitfly	1.000	$=$	1.000	$1.1 \cdot 10^2$	$=$	$9.0 \cdot 10^1$	$0.0 \cdot 10^0$	$=$	$0.0 \cdot 10^0$
house-8l	0.734	$<$	0.794	$5.1 \cdot 10^3$	$>$	$6.0 \cdot 10^2$	$9.0 \cdot 10^1$	$=$	$3.0 \cdot 10^1$
housing	0.484	$=$	0.576	$2.0 \cdot 10^4$	$>$	$1.1 \cdot 10^3$	$1.0 \cdot 10^2$	$>$	$3.0 \cdot 10^1$
kin8nm	0.792	$=$	0.791	$5.1 \cdot 10^3$	$=$	$6.0 \cdot 10^2$	$9.0 \cdot 10^1$	$=$	$4.0 \cdot 10^1$
lowbwt	0.626	$=$	0.626	$1.8 \cdot 10^4$	$>$	$1.3 \cdot 10^3$	$1.1 \cdot 10^1$	$=$	$1.1 \cdot 10^1$
mbagrade	1.040	$=$	1.040	$5.2 \cdot 10^1$	$>$	$3.3 \cdot 10^1$	$2.0 \cdot 10^0$	$=$	$2.0 \cdot 10^0$
meta	0.988	$=$	1.000	$6.4 \cdot 10^5$	$>$	$5.1 \cdot 10^3$	$2.5 \cdot 10^2$	$>$	$2.8 \cdot 10^1$
mv	0.116	$<$	0.435	$1.0 \cdot 10^4$	$=$	$6.2 \cdot 10^2$	$1.0 \cdot 10^2$	$=$	$3.0 \cdot 10^1$
pharynx	0.740	$=$	0.733	$6.2 \cdot 10^4$	$>$	$1.3 \cdot 10^3$	$3.1 \cdot 10^1$	$>$	$2.2 \cdot 10^1$
puma32h	0.884	$=$	0.884	$1.6 \cdot 10^3$	$=$	$6.4 \cdot 10^2$	$1.0 \cdot 10^1$	$=$	$1.0 \cdot 10^1$
pw-linear	0.583	$=$	0.623	$1.0 \cdot 10^4$	$>$	$6.2 \cdot 10^2$	$1.0 \cdot 10^2$	$>$	$3.0 \cdot 10^1$
pyrim	0.972	$=$	0.951	$7.1 \cdot 10^4$	$>$	$9.0 \cdot 10^2$	$3.8 \cdot 10^1$	$>$	$2.0 \cdot 10^1$
quake	1.000	$=$	1.000	$6.0 \cdot 10^2$	$=$	$8.0 \cdot 10^1$	$0.0 \cdot 10^0$	$=$	$0.0 \cdot 10^0$
sensory	0.919	$<$	0.957	$8.8 \cdot 10^4$	$>$	$1.5 \cdot 10^3$	$5.6 \cdot 10^1$	$>$	$1.8 \cdot 10^1$
servo	0.458	$<$	0.572	$3.5 \cdot 10^4$	$>$	$1.2 \cdot 10^3$	$1.1 \cdot 10^2$	$>$	$3.0 \cdot 10^1$
strike	0.932	$=$	0.937	$7.8 \cdot 10^4$	$>$	$1.3 \cdot 10^3$	$6.4 \cdot 10^1$	$>$	$3.0 \cdot 10^1$
triazines	0.999	$=$	0.999	$1.7 \cdot 10^3$	$=$	$6.8 \cdot 10^2$	$2.0 \cdot 10^0$	$=$	$2.0 \cdot 10^0$
veteran	0.944	$=$	0.941	$8.1 \cdot 10^3$	$>$	$6.1 \cdot 10^2$	$1.9 \cdot 10^1$	$=$	$1.5 \cdot 10^1$
vineyard	0.780	$=$	0.702	$5.4 \cdot 10^2$	$>$	$8.0 \cdot 10^1$	$3.1 \cdot 10^1$	$>$	$1.2 \cdot 10^1$

Table 51: The performance of CIPER using the new refinement operator and the CV heuristic when learning quadratic equations - with a maximum degree of 2 (Q) and linear equations with a maximum degree of 1 (L).

	rrmse			*ssc*			*mcl*		
dataset	Q	t-test	L	Q	t-test	L	Q	t-test	L
2dplanes	0.482	$<$	0.621	$1.0 \cdot 10^5$	$>$	$6.2 \cdot 10^3$	$3.5 \cdot 10^2$	$>$	$3.0 \cdot 10^1$
auto-price	0.397	$<$	0.456	$3.1 \cdot 10^5$	$>$	$9.9 \cdot 10^3$	$6.6 \cdot 10^2$	$>$	$1.1 \cdot 10^2$
bank32nh	0.672	$<$	0.749	$3.0 \cdot 10^6$	$>$	$2.5 \cdot 10^4$	$1.3 \cdot 10^3$	$>$	$6.1 \cdot 10^1$
baskball	0.824	$=$	0.824	$2.2 \cdot 10^3$	$>$	$6.6 \cdot 10^2$	$8.4 \cdot 10^1$	$>$	$2.9 \cdot 10^1$
bodyfat	0.164	$=$	0.160	$1.4 \cdot 10^5$	$>$	$8.4 \cdot 10^3$	$3.2 \cdot 10^2$	$>$	$8.5 \cdot 10^1$
cal-housing	0.643	$<$	0.646	$5.1 \cdot 10^4$	$>$	$6.0 \cdot 10^3$	$1.2 \cdot 10^2$	$>$	$3.0 \cdot 10^1$
cholesterol	0.992	$=$	0.981	$5.9 \cdot 10^5$	$>$	$1.0 \cdot 10^4$	$7.3 \cdot 10^2$	$>$	$1.0 \cdot 10^2$
cloud	0.399	$=$	0.414	$6.3 \cdot 10^4$	$>$	$2.7 \cdot 10^3$	$2.8 \cdot 10^2$	$>$	$4.6 \cdot 10^1$
cpu-small	0.217	$<$	0.551	$2.0 \cdot 10^5$	$>$	$1.1 \cdot 10^4$	$3.4 \cdot 10^2$	$>$	$5.6 \cdot 10^1$
delta-ailerons	0.586	$>$	0.569	$1.3 \cdot 10^4$	$>$	$3.0 \cdot 10^3$	$8.4 \cdot 10^1$	$>$	$4.8 \cdot 10^1$
delta-elevators	0.648	$>$	0.611	$2.6 \cdot 10^4$	$>$	$3.4 \cdot 10^3$	$1.6 \cdot 10^2$	$>$	$5.6 \cdot 10^1$
elevators	0.423	$<$	0.540	$2.4 \cdot 10^5$	$>$	$1.2 \cdot 10^4$	$4.4 \cdot 10^2$	$>$	$4.8 \cdot 10^1$
elusage	0.447	$=$	0.477	$9.9 \cdot 10^2$	$>$	$3.6 \cdot 10^2$	$3.2 \cdot 10^1$	$>$	$1.0 \cdot 10^1$
fried-delve	0.537	$<$	0.598	$1.0 \cdot 10^5$	$>$	$6.3 \cdot 10^3$	$4.0 \cdot 10^2$	$>$	$4.7 \cdot 10^1$
fruitfly	1.009	$=$	1.010	$1.5 \cdot 10^3$	$=$	$8.8 \cdot 10^2$	$1.3 \cdot 10^1$	$=$	$8.0 \cdot 10^0$
house-8l	0.717	$<$	0.793	$5.1 \cdot 10^4$	$>$	$6.0 \cdot 10^3$	$2.1 \cdot 10^2$	$>$	$5.1 \cdot 10^1$
housing	0.455	$<$	0.566	$2.0 \cdot 10^5$	$>$	$1.1 \cdot 10^4$	$5.5 \cdot 10^2$	$>$	$9.2 \cdot 10^1$
kin8nm	0.782	$=$	0.788	$5.1 \cdot 10^4$	$>$	$6.1 \cdot 10^3$	$2.3 \cdot 10^2$	$>$	$6.3 \cdot 10^1$
lowbwt	0.607	$=$	0.608	$2.9 \cdot 10^5$	$>$	$1.0 \cdot 10^4$	$5.2 \cdot 10^2$	$>$	$7.6 \cdot 10^1$
mbagrade	0.929	$=$	0.940	$5.5 \cdot 10^2$	$>$	$3.2 \cdot 10^2$	$4.8 \cdot 10^1$	$>$	$2.0 \cdot 10^1$
meta	1.002	$=$	0.977	$3.1 \cdot 10^6$	$>$	$4.8 \cdot 10^4$	$6.7 \cdot 10^2$	$>$	$6.9 \cdot 10^1$
mv	0.117	$<$	0.433	$1.0 \cdot 10^5$	$>$	$7.8 \cdot 10^3$	$2.9 \cdot 10^2$	$>$	$8.6 \cdot 10^1$
pharynx	0.716	$=$	0.724	$7.9 \cdot 10^5$	$>$	$10.0 \cdot 10^3$	$5.7 \cdot 10^2$	$>$	$6.6 \cdot 10^1$
puma32h	0.885	$=$	0.884	$1.7 \cdot 10^6$	$>$	$2.6 \cdot 10^4$	$8.4 \cdot 10^2$	$>$	$8.1 \cdot 10^1$
pw-linear	0.555	$<$	0.588	$7.0 \cdot 10^4$	$>$	$4.2 \cdot 10^3$	$3.6 \cdot 10^2$	$>$	$7.2 \cdot 10^1$
pyrim	0.767	$=$	0.807	$5.6 \cdot 10^5$	$>$	$8.3 \cdot 10^3$	$5.0 \cdot 10^2$	$>$	$1.1 \cdot 10^2$
quake	0.998	$=$	0.998	$6.6 \cdot 10^3$	$>$	$8.0 \cdot 10^2$	$1.2 \cdot 10^2$	$>$	$2.4 \cdot 10^1$
sensory	0.874	$<$	0.949	$1.2 \cdot 10^6$	$>$	$1.1 \cdot 10^4$	$7.5 \cdot 10^2$	$>$	$8.6 \cdot 10^1$
servo	0.498	$<$	0.632	$2.3 \cdot 10^5$	$>$	$8.6 \cdot 10^3$	$3.7 \cdot 10^2$	$>$	$1.0 \cdot 10^2$
strike	0.910	$<$	0.931	$6.3 \cdot 10^5$	$>$	$1.3 \cdot 10^4$	$6.9 \cdot 10^2$	$>$	$9.3 \cdot 10^1$
triazines	0.871	$<$	0.951	$2.1 \cdot 10^6$	$>$	$1.7 \cdot 10^4$	$7.1 \cdot 10^2$	$>$	$9.8 \cdot 10^1$
veteran	0.900	$=$	0.917	$3.3 \cdot 10^4$	$>$	$2.4 \cdot 10^3$	$2.2 \cdot 10^2$	$>$	$5.1 \cdot 10^1$
vineyard	0.637	$=$	0.670	$2.7 \cdot 10^3$	$>$	$5.6 \cdot 10^2$	$1.1 \cdot 10^2$	$>$	$2.4 \cdot 10^1$

Table 52: The performance of CIPER using the improved MDL heuristic when learning polynomial equations - CIPER without a limitation on the degree (P) and quadratic equations - CIPER with a maximum degree of 2 (Q).

	rrmse			*ssc*			*mcl*		
dataset	P	t-test	Q	P	t-test	Q	P	t-test	Q
2dplanes	0.227	<	0.562	$8.9 \cdot 10^5$	>	$1.0 \cdot 10^4$	$2.3 \cdot 10^2$	>	$1.1 \cdot 10^2$
auto-price	0.396	<	0.458	$1.7 \cdot 10^5$	>	$3.0 \cdot 10^4$	$5.8 \cdot 10^1$	>	$4.1 \cdot 10^1$
bank32nh	0.655	<	0.671	$4.0 \cdot 10^7$	>	$3.2 \cdot 10^5$	$1.1 \cdot 10^3$	>	$3.3 \cdot 10^2$
baskball	0.774	=	0.774	$2.6 \cdot 10^3$	=	$5.4 \cdot 10^2$	$2.0 \cdot 10^1$	=	$2.0 \cdot 10^1$
bodyfat	0.170	=	0.160	$1.5 \cdot 10^5$	>	$2.0 \cdot 10^4$	$4.7 \cdot 10^1$	=	$1.2 \cdot 10^1$
cal-housing	0.829	=	0.648	$3.1 \cdot 10^6$	>	$5.1 \cdot 10^3$	$1.2 \cdot 10^3$	>	$5.8 \cdot 10^1$
cholesterol	1.008	=	1.008	$5.0 \cdot 10^4$	>	$2.5 \cdot 10^4$	$1.0 \cdot 10^0$	=	$1.0 \cdot 10^0$
cloud	0.437	=	0.383	$9.6 \cdot 10^4$	>	$9.2 \cdot 10^3$	$4.5 \cdot 10^1$	=	$2.9 \cdot 10^1$
cpu-small	0.174	<	0.221	$5.7 \cdot 10^6$	>	$2.0 \cdot 10^4$	$8.8 \cdot 10^2$	>	$1.3 \cdot 10^2$
delta-ailerons	0.569	<	1.000	$1.5 \cdot 10^4$	=	$1.1 \cdot 10^2$	$3.0 \cdot 10^1$	=	$0.0 \cdot 10^0$
delta-elevators	0.631	<	1.000	$2.2 \cdot 10^3$	=	$1.3 \cdot 10^2$	$2.0 \cdot 10^1$	=	$0.0 \cdot 10^0$
elevators	0.355	<	1.000	$1.4 \cdot 10^6$	>	$3.3 \cdot 10^2$	$2.7 \cdot 10^2$	>	$0.0 \cdot 10^0$
elusage	0.401	=	0.401	$3.1 \cdot 10^2$	=	$1.4 \cdot 10^2$	$3.0 \cdot 10^1$	=	$3.0 \cdot 10^1$
fried-delve	0.201	<	0.566	$9.0 \cdot 10^5$	>	$1.0 \cdot 10^4$	$3.3 \cdot 10^2$	>	$1.1 \cdot 10^2$
fruitfly	1.000	=	1.000	$1.1 \cdot 10^2$	=	$1.1 \cdot 10^2$	$0.0 \cdot 10^0$	=	$0.0 \cdot 10^0$
house-8l	0.620	<	0.734	$2.1 \cdot 10^6$	>	$5.1 \cdot 10^3$	$1.1 \cdot 10^3$	>	$9.0 \cdot 10^1$
housing	0.397	<	0.484	$1.4 \cdot 10^6$	>	$2.0 \cdot 10^4$	$2.9 \cdot 10^2$	>	$1.0 \cdot 10^2$
kin8nm	0.500	<	0.792	$2.4 \cdot 10^6$	>	$5.1 \cdot 10^3$	$1.0 \cdot 10^3$	>	$9.0 \cdot 10^1$
lowbwt	0.622	=	0.626	$4.2 \cdot 10^4$	>	$1.8 \cdot 10^4$	$1.0 \cdot 10^1$	=	$1.1 \cdot 10^1$
mbagrade	1.040	=	1.040	$5.6 \cdot 10^1$	=	$5.2 \cdot 10^1$	$2.0 \cdot 10^0$	=	$2.0 \cdot 10^0$
meta	1.052	=	0.988	$2.4 \cdot 10^7$	>	$6.4 \cdot 10^5$	$6.2 \cdot 10^2$	>	$2.5 \cdot 10^2$
mv	0.029	<	0.116	$6.2 \cdot 10^5$	>	$1.0 \cdot 10^4$	$3.5 \cdot 10^2$	>	$1.0 \cdot 10^2$
pharynx	0.743	=	0.740	$1.6 \cdot 10^5$	>	$6.2 \cdot 10^4$	$3.1 \cdot 10^1$	=	$3.1 \cdot 10^1$
puma32h	0.271	<	0.884	$5.0 \cdot 10^3$	=	$1.6 \cdot 10^3$	$4.0 \cdot 10^1$	=	$1.0 \cdot 10^1$
pw-linear	0.420	<	0.583	$2.3 \cdot 10^5$	>	$1.0 \cdot 10^4$	$1.6 \cdot 10^2$	=	$1.0 \cdot 10^2$
pyrim	0.829	=	0.972	$2.6 \cdot 10^5$	>	$7.1 \cdot 10^4$	$4.0 \cdot 10^1$	=	$3.8 \cdot 10^1$
quake	1.000	=	1.000	$5.0 \cdot 10^3$	>	$6.0 \cdot 10^2$	$0.0 \cdot 10^0$	=	$0.0 \cdot 10^0$
sensory	0.925	=	0.919	$2.0 \cdot 10^5$	>	$8.8 \cdot 10^4$	$4.7 \cdot 10^1$	=	$5.6 \cdot 10^1$
servo	0.302	<	0.458	$6.1 \cdot 10^5$	>	$3.5 \cdot 10^4$	$3.0 \cdot 10^2$	>	$1.1 \cdot 10^2$
strike	0.936	=	0.932	$5.1 \cdot 10^5$	>	$7.8 \cdot 10^4$	$8.5 \cdot 10^1$	>	$6.4 \cdot 10^1$
triazines	0.999	=	0.999	$2.0 \cdot 10^3$	=	$1.7 \cdot 10^3$	$2.0 \cdot 10^0$	=	$2.0 \cdot 10^0$
veteran	1.476	=	0.944	$7.3 \cdot 10^4$	>	$8.1 \cdot 10^3$	$5.8 \cdot 10^1$	=	$1.9 \cdot 10^1$
vineyard	0.701	=	0.780	$8.2 \cdot 10^3$	>	$5.4 \cdot 10^2$	$3.5 \cdot 10^1$	=	$3.1 \cdot 10^1$

Table 53: The performance of CIPER using the CV heuristic when learning polynomial equations - CIPER without a limitation on the degree (P) and quadratic equations - CIPER with a maximum degree of 2 (Q).

	rrmse			*ssc*			*mcl*		
dataset	P	t-test	Q	P	t-test	Q	P	t-test	Q
2dplanes	0.228	$<$	0.503	$1.4 \cdot 10^7$	$>$	$1.0 \cdot 10^5$	$2.6 \cdot 10^3$	$>$	$3.5 \cdot 10^2$
auto-price	0.372	$=$	0.394	$2.8 \cdot 10^6$	$>$	$3.1 \cdot 10^5$	$2.2 \cdot 10^3$	$>$	$6.6 \cdot 10^2$
bank32nh	0.670	$=$	0.673	$5.4 \cdot 10^7$	$>$	$3.0 \cdot 10^6$	$3.0 \cdot 10^3$	$>$	$1.3 \cdot 10^3$
baskball	0.829	$=$	0.821	$2.7 \cdot 10^4$	$>$	$2.2 \cdot 10^3$	$2.6 \cdot 10^2$	$>$	$8.4 \cdot 10^1$
bodyfat	0.166	$=$	0.167	$7.8 \cdot 10^5$	$>$	$1.4 \cdot 10^5$	$5.8 \cdot 10^2$	$=$	$3.2 \cdot 10^2$
cal-housing	0.551	$<$	0.644	$1.3 \cdot 10^7$	$>$	$5.1 \cdot 10^4$	$4.8 \cdot 10^3$	$>$	$1.2 \cdot 10^2$
cholesterol	1.025	$=$	0.980	$5.2 \cdot 10^6$	$>$	$5.9 \cdot 10^5$	$2.4 \cdot 10^3$	$>$	$7.3 \cdot 10^2$
cloud	0.507	$=$	0.436	$8.6 \cdot 10^5$	$>$	$6.3 \cdot 10^4$	$1.1 \cdot 10^3$	$>$	$2.8 \cdot 10^2$
cpu-small	0.189	$<$	0.217	$1.0 \cdot 10^7$	$>$	$2.0 \cdot 10^5$	$1.3 \cdot 10^3$	$>$	$3.4 \cdot 10^2$
delta-ailerons	0.550	$<$	0.569	$4.3 \cdot 10^6$	$>$	$1.3 \cdot 10^4$	$3.0 \cdot 10^3$	$>$	$8.4 \cdot 10^1$
delta-elevators	0.603	$<$	0.641	$3.3 \cdot 10^6$	$>$	$2.6 \cdot 10^4$	$1.7 \cdot 10^3$	$>$	$1.6 \cdot 10^2$
elevators	0.367	$=$	0.366	$9.5 \cdot 10^6$	$>$	$2.4 \cdot 10^5$	$1.3 \cdot 10^3$	$>$	$4.4 \cdot 10^2$
elusage	0.453	$=$	0.446	$1.4 \cdot 10^3$	$>$	$9.9 \cdot 10^2$	$5.7 \cdot 10^1$	$>$	$3.2 \cdot 10^1$
fried-delve	0.201	$<$	0.528	$1.0 \cdot 10^7$	$>$	$1.0 \cdot 10^5$	$2.5 \cdot 10^3$	$>$	$4.0 \cdot 10^2$
fruitfly	1.010	$=$	1.000	$5.8 \cdot 10^3$	$=$	$1.5 \cdot 10^3$	$2.8 \cdot 10^1$	$=$	$1.3 \cdot 10^1$
house-8l	0.611	$<$	0.714	$1.3 \cdot 10^7$	$>$	$5.1 \cdot 10^4$	$4.7 \cdot 10^3$	$>$	$2.1 \cdot 10^2$
housing	0.406	$=$	0.427	$5.0 \cdot 10^6$	$>$	$2.0 \cdot 10^5$	$2.4 \cdot 10^3$	$>$	$5.5 \cdot 10^2$
kin8nm	0.579	$<$	0.777	$7.8 \cdot 10^6$	$>$	$5.1 \cdot 10^4$	$1.0 \cdot 10^3$	$>$	$2.3 \cdot 10^2$
lowbwt	0.601	$=$	0.593	$2.8 \cdot 10^6$	$>$	$2.9 \cdot 10^5$	$1.7 \cdot 10^3$	$>$	$5.2 \cdot 10^2$
mbagrade	0.938	$=$	0.925	$5.5 \cdot 10^2$	$=$	$5.5 \cdot 10^2$	$4.4 \cdot 10^1$	$=$	$4.8 \cdot 10^1$
meta	1.500	$=$	0.993	$2.2 \cdot 10^7$	$>$	$3.1 \cdot 10^6$	$2.5 \cdot 10^3$	$>$	$6.7 \cdot 10^2$
mv	0.026	$<$	0.049	$7.5 \cdot 10^6$	$>$	$1.0 \cdot 10^5$	$1.7 \cdot 10^3$	$>$	$2.9 \cdot 10^2$
pharynx	0.729	$=$	0.716	$2.8 \cdot 10^6$	$>$	$7.9 \cdot 10^5$	$8.6 \cdot 10^2$	$>$	$5.7 \cdot 10^2$
puma32h	0.271	$<$	0.884	$1.8 \cdot 10^7$	$>$	$1.7 \cdot 10^6$	$1.8 \cdot 10^3$	$=$	$8.4 \cdot 10^2$
pw-linear	0.483	$<$	0.559	$1.2 \cdot 10^6$	$>$	$7.0 \cdot 10^4$	$5.6 \cdot 10^2$	$>$	$3.6 \cdot 10^2$
pyrim	0.728	$=$	0.625	$2.2 \cdot 10^6$	$>$	$5.6 \cdot 10^5$	$1.0 \cdot 10^3$	$>$	$5.0 \cdot 10^2$
quake	0.997	$=$	0.997	$6.8 \cdot 10^5$	$>$	$6.6 \cdot 10^3$	$1.7 \cdot 10^3$	$>$	$1.2 \cdot 10^2$
sensory	0.868	$=$	0.859	$5.6 \cdot 10^6$	$>$	$1.2 \cdot 10^6$	$10.0 \cdot 10^2$	$>$	$7.5 \cdot 10^2$
servo	0.483	$>$	0.402	$1.0 \cdot 10^6$	$>$	$2.3 \cdot 10^5$	$4.3 \cdot 10^2$	$=$	$3.7 \cdot 10^2$
strike	0.914	$=$	0.911	$4.0 \cdot 10^6$	$>$	$6.3 \cdot 10^5$	$1.2 \cdot 10^3$	$>$	$6.9 \cdot 10^2$
triazines	1.152	$=$	0.867	$7.7 \cdot 10^6$	$>$	$2.1 \cdot 10^6$	$1.4 \cdot 10^3$	$>$	$7.1 \cdot 10^2$
veteran	0.901	$=$	0.890	$3.5 \cdot 10^5$	$>$	$3.3 \cdot 10^4$	$5.1 \cdot 10^2$	$>$	$2.2 \cdot 10^2$
vineyard	0.632	$=$	0.650	$3.8 \cdot 10^4$	$>$	$2.7 \cdot 10^3$	$4.3 \cdot 10^2$	$>$	$1.1 \cdot 10^2$

A.6 CIPER vs LR, RT, and MT

Table 54: The predictive performance (in terms of *rrmse*) of MDLCIPER (CIPER with the MDL heuristic) as compared to linear regression (**LR**), regression trees (**RT**), and model trees (**MT**). The algorithms LR, RT, and MT are used as implemented in the WEKA data mining software [24].

dataset	MDLCIPER	t-test	**LR**	t-test	**RT**	t-test	**MT**
2dplanes	0.227	<	0.543	=	0.227	=	0.227
auto-price	0.396	=	0.484	=	0.564	=	0.379
bank32nh	0.655	=	0.685	<	0.752	=	0.673
baskball	0.774	=	0.790	=	0.882	=	0.790
bodyfat	0.170	=	0.165	<	0.329	=	0.158
cal-housing	0.829	=	0.603	=	0.515	=	0.486
cholesterol	1.008	=	1.001	=	1.007	=	1.023
cloud	0.437	=	0.387	<	0.738	=	0.381
cpu-small	0.174	<	0.537	<	0.223	=	0.173
delta-ailerons	0.569	=	0.568	=	0.577	=	0.544
delta-elevators	0.631	>	0.610	=	0.622	>	0.600
elevators	0.355	<	0.433	<	0.521	>	0.322
elusage	0.401	=	0.472	=	0.657	=	0.413
fried-delve	0.201	<	0.527	<	0.357	<	0.278
fruitfly	1.000	=	1.000	=	1.000	=	1.000
house-8l	0.620	<	0.788	=	0.622	=	0.594
housing	0.397	<	0.533	<	0.523	=	0.407
kin8nm	0.500	<	0.766	<	0.688	<	0.607
lowbwt	0.622	=	0.605	=	0.640	=	0.609
mbagrade	1.040	=	0.891	=	1.000	=	0.891
meta	1.052	=	0.992	=	0.992	=	1.031
mv	0.029	<	0.431	<	0.048	=	0.013
pharynx	0.743	<	1.190	<	1.095	<	1.053
puma32h	0.271	<	0.885	<	0.290	=	0.270
pw-linear	0.420	=	0.505	<	0.569	=	0.324
pyrim	0.829	=	0.932	=	1.023	=	0.695
quake	1.000	=	0.998	=	1.001	=	0.996
sensory	0.925	=	0.925	=	0.911	>	0.861
servo	0.302	<	0.541	<	0.628	=	0.352
strike	0.936	>	0.912	=	0.942	=	0.910
triazines	0.999	=	0.968	>	0.901	=	0.849
veteran	1.476	=	0.903	=	0.950	=	0.897
vineyard	0.701	=	0.665	=	0.832	=	0.674

Table 55: The predictive performance of CVCiper (Ciper with the CV heuristic) as compared to linear regression (**LR**), regression trees (**RT**), and model trees (**MT**). The algorithms LR, RT, and MT are used as implemented in the WEKA data mining software [24].

dataset	CVCiper	t-test	**LR**	t-test	**RT**	t-test	**MT**
2dplanes	0.228	<	0.543	=	0.227	=	0.227
auto-price	0.372	=	0.484	=	0.564	=	0.379
bank32nh	0.670	=	0.685	<	0.752	=	0.673
baskball	0.829	=	0.790	=	0.882	=	0.790
bodyfat	0.166	=	0.165	<	0.329	=	0.158
cal-housing	0.551	<	0.603	>	0.515	>	0.486
cholesterol	1.025	=	1.001	=	1.007	=	1.023
cloud	0.507	=	0.387	<	0.738	=	0.381
cpu-small	0.189	<	0.537	<	0.223	=	0.173
delta-ailerons	0.550	=	0.568	=	0.577	=	0.544
delta-elevators	0.603	=	0.610	=	0.622	=	0.600
elevators	0.367	<	0.433	<	0.521	>	0.322
elusage	0.453	=	0.472	=	0.657	=	0.413
fried-delve	0.201	<	0.527	<	0.357	<	0.278
fruitfly	1.010	=	1.000	=	1.000	=	1.000
house-8l	0.611	<	0.788	=	0.622	=	0.594
housing	0.406	<	0.533	<	0.523	=	0.407
kin8nm	0.579	<	0.766	<	0.688	<	0.607
lowbwt	0.601	=	0.605	=	0.640	=	0.609
mbagrade	0.938	=	0.891	=	1.000	=	0.891
meta	1.500	=	0.992	=	0.992	=	1.031
mv	0.026	<	0.431	<	0.048	=	0.013
pharynx	0.729	<	1.190	<	1.095	<	1.053
puma32h	0.271	<	0.885	<	0.290	=	0.270
pw-linear	0.483	=	0.505	=	0.569	>	0.324
pyrim	0.728	=	0.932	<	1.023	=	0.695
quake	0.997	=	0.998	=	1.001	=	0.996
sensory	0.868	<	0.925	<	0.911	=	0.861
servo	0.483	=	0.541	=	0.628	>	0.352
strike	0.914	=	0.912	=	0.942	=	0.910
triazines	1.152	=	0.968	=	0.901	=	0.849
veteran	0.901	=	0.903	=	0.950	=	0.897
vineyard	0.632	=	0.665	=	0.832	=	0.674

B Evaluating CIPER Extensions: Complete Results

B.1 Evaluating Piecewise Polynomial Models

Table 56: The performance of polynomial equations learned by MDLCIPER (P) compared to the performance of piecewise linear equations learned by MDLCIPERX - with a maximum non-binary degree variables of 1 (Lx).

	rrmse			*ssc*			*mcl*		
dataset	P	t-test	Lx	P	t-test	Lx	P	t-test	Lx
2dplanes	0.227	=	0.227	$8.9 \cdot 10^5$	<	$1.6 \cdot 10^6$	$2.3 \cdot 10^2$	>	$1.9 \cdot 10^2$
auto-price	0.396	<	0.462	$1.7 \cdot 10^5$	>	$2.9 \cdot 10^4$	$5.8 \cdot 10^1$	>	$2.7 \cdot 10^1$
bank32nh	0.655	<	0.678	$4.0 \cdot 10^7$	>	$1.6 \cdot 10^7$	$1.1 \cdot 10^3$	>	$3.5 \cdot 10^2$
baskball	0.774	=	0.774	$2.6 \cdot 10^3$	=	$5.8 \cdot 10^2$	$2.0 \cdot 10^1$	=	$2.0 \cdot 10^1$
bodyfat	0.170	=	0.162	$1.5 \cdot 10^5$	>	$1.1 \cdot 10^3$	$4.7 \cdot 10^1$	=	$1.9 \cdot 10^1$
cal-housing	0.829	=	0.595	$3.1 \cdot 10^6$	>	$2.7 \cdot 10^5$	$1.2 \cdot 10^3$	>	$3.0 \cdot 10^2$
cholesterol	1.008	=	1.008	$5.0 \cdot 10^4$	>	$4.9 \cdot 10^2$	$1.0 \cdot 10^0$	=	$1.0 \cdot 10^0$
cloud	0.437	=	0.375	$9.6 \cdot 10^4$	>	$1.4 \cdot 10^3$	$4.5 \cdot 10^1$	=	$2.8 \cdot 10^1$
cpu-small	0.174	=	1.473	$5.7 \cdot 10^6$	>	$1.5 \cdot 10^6$	$8.8 \cdot 10^2$	>	$3.5 \cdot 10^2$
delta-ailerons	0.569	=	0.569	$1.5 \cdot 10^4$	=	$4.0 \cdot 10^3$	$3.0 \cdot 10^1$	=	$3.0 \cdot 10^1$
delta-elevators	0.631	>	0.615	$2.2 \cdot 10^3$	<	$6.0 \cdot 10^3$	$2.0 \cdot 10^1$	<	$4.5 \cdot 10^1$
elevators	0.355	<	0.430	$1.4 \cdot 10^6$	>	$1.2 \cdot 10^5$	$2.7 \cdot 10^2$	>	$1.9 \cdot 10^2$
elusage	0.401	=	0.478	$3.1 \cdot 10^2$	=	$3.2 \cdot 10^2$	$3.0 \cdot 10^1$	=	$1.0 \cdot 10^1$
fried-delve	0.201	<	0.376	$9.0 \cdot 10^5$	<	$1.2 \cdot 10^7$	$3.3 \cdot 10^2$	<	$6.1 \cdot 10^2$
fruitfly	1.000	=	1.000	$1.1 \cdot 10^2$	=	$9.0 \cdot 10^1$	$0.0 \cdot 10^0$	=	$0.0 \cdot 10^0$
house-8l	0.620	<	0.761	$2.1 \cdot 10^6$	=	$1.8 \cdot 10^6$	$1.1 \cdot 10^3$	>	$5.6 \cdot 10^2$
housing	0.397	=	0.460	$1.4 \cdot 10^6$	>	$8.2 \cdot 10^5$	$2.9 \cdot 10^2$	>	$1.8 \cdot 10^2$
kin8nm	0.500	<	0.663	$2.4 \cdot 10^6$	=	$7.7 \cdot 10^6$	$1.0 \cdot 10^3$	=	$6.4 \cdot 10^2$
lowbwt	0.622	=	0.626	$4.2 \cdot 10^4$	>	$8.6 \cdot 10^2$	$1.0 \cdot 10^1$	=	$1.1 \cdot 10^1$
mbagrade	1.040	=	1.040	$5.6 \cdot 10^1$	=	$3.4 \cdot 10^1$	$2.0 \cdot 10^0$	=	$2.0 \cdot 10^0$
meta	1.052	=	1.410	$2.4 \cdot 10^7$	=	$2.1 \cdot 10^7$	$6.2 \cdot 10^2$	>	$4.4 \cdot 10^2$
mv	0.029	>	0.026	$6.2 \cdot 10^5$	<	$2.1 \cdot 10^6$	$3.5 \cdot 10^2$	=	$3.3 \cdot 10^2$
pharynx	0.743	=	0.746	$1.6 \cdot 10^5$	=	$1.3 \cdot 10^5$	$3.1 \cdot 10^1$	=	$2.6 \cdot 10^1$
puma32h	0.271	<	0.718	$5.0 \cdot 10^3$	=	$1.3 \cdot 10^6$	$4.0 \cdot 10^1$	=	$2.3 \cdot 10^1$
pw-linear	0.420	=	0.348	$2.3 \cdot 10^5$	<	$5.3 \cdot 10^5$	$1.6 \cdot 10^2$	=	$1.5 \cdot 10^2$
pyrim	0.829	=	1.500	$2.6 \cdot 10^5$	=	$3.1 \cdot 10^5$	$4.0 \cdot 10^1$	=	$3.0 \cdot 10^1$
quake	1.000	=	1.000	$5.0 \cdot 10^3$	>	$1.1 \cdot 10^2$	$0.0 \cdot 10^0$	=	$0.0 \cdot 10^0$
sensory	0.925	=	0.955	$2.0 \cdot 10^5$	>	$1.5 \cdot 10^4$	$4.7 \cdot 10^1$	>	$2.0 \cdot 10^1$
servo	0.302	=	0.309	$6.1 \cdot 10^5$	>	$5.3 \cdot 10^5$	$3.0 \cdot 10^2$	=	$3.0 \cdot 10^2$
strike	0.936	=	0.933	$5.1 \cdot 10^5$	>	$2.7 \cdot 10^5$	$8.5 \cdot 10^1$	=	$7.7 \cdot 10^1$
triazines	0.999	=	0.999	$2.0 \cdot 10^3$	<	$3.2 \cdot 10^3$	$2.0 \cdot 10^0$	=	$2.0 \cdot 10^0$
veteran	1.476	=	0.950	$7.3 \cdot 10^4$	>	$9.3 \cdot 10^2$	$5.8 \cdot 10^1$	=	$1.4 \cdot 10^1$
vineyard	0.701	=	0.702	$8.2 \cdot 10^3$	>	$7.7 \cdot 10^1$	$3.5 \cdot 10^1$	>	$1.2 \cdot 10^1$

Table 57: The performance of polynomial equations learned by CVCIPER (P) compared to the performance of piecewise linear equations learned by CVCIPERX - with a maximum non-binary degree variables of 1 (Lx).

	rrmse			*ssc*			*mcl*		
dataset	P	t-test	Lx	P	t-test	Lx	P	t-test	Lx
2dplanes	0.228	=	0.227	$1.4 \cdot 10^7$	<	$1.9 \cdot 10^7$	$2.6 \cdot 10^3$	>	$1.3 \cdot 10^3$
auto-price	0.372	=	0.438	$2.8 \cdot 10^6$	>	$4.8 \cdot 10^5$	$2.2 \cdot 10^3$	>	$5.5 \cdot 10^2$
bank32nh	0.670	<	0.685	$5.4 \cdot 10^7$	=	$6.2 \cdot 10^7$	$3.0 \cdot 10^3$	>	$1.5 \cdot 10^3$
baskball	0.829	=	0.810	$2.7 \cdot 10^4$	>	$3.1 \cdot 10^3$	$2.6 \cdot 10^2$	>	$6.9 \cdot 10^1$
bodyfat	0.166	=	0.160	$7.8 \cdot 10^5$	>	$8.2 \cdot 10^3$	$5.8 \cdot 10^2$	>	$9.2 \cdot 10^1$
cal-housing	0.551	<	0.599	$1.3 \cdot 10^7$	>	$1.1 \cdot 10^6$	$4.8 \cdot 10^3$	>	$8.3 \cdot 10^2$
cholesterol	1.025	=	0.984	$5.2 \cdot 10^6$	=	$4.0 \cdot 10^6$	$2.4 \cdot 10^3$	>	$9.6 \cdot 10^2$
cloud	0.507	=	0.416	$8.6 \cdot 10^5$	>	$3.1 \cdot 10^5$	$1.1 \cdot 10^3$	>	$4.3 \cdot 10^2$
cpu-small	0.189	>	0.176	$1.0 \cdot 10^7$	>	$6.2 \cdot 10^6$	$1.3 \cdot 10^3$	=	$1.3 \cdot 10^3$
delta-ailerons	0.550	<	0.567	$4.3 \cdot 10^6$	>	$4.2 \cdot 10^4$	$3.0 \cdot 10^3$	>	$1.3 \cdot 10^2$
delta-elevators	0.603	<	0.614	$3.3 \cdot 10^6$	>	$2.3 \cdot 10^5$	$1.7 \cdot 10^3$	>	$2.9 \cdot 10^2$
elevators	0.367	<	0.436	$9.5 \cdot 10^6$	>	$8.6 \cdot 10^5$	$1.3 \cdot 10^3$	>	$5.8 \cdot 10^2$
elusage	0.453	=	0.455	$1.4 \cdot 10^3$	=	$10.0 \cdot 10^3$	$5.7 \cdot 10^1$	<	$1.1 \cdot 10^2$
fried-delve	0.201	<	0.366	$1.0 \cdot 10^7$	<	$6.2 \cdot 10^7$	$2.5 \cdot 10^3$	=	$2.3 \cdot 10^3$
fruitfly	1.010	=	1.000	$5.8 \cdot 10^3$	=	$8.0 \cdot 10^2$	$2.8 \cdot 10^1$	=	$0.0 \cdot 10^0$
house-8l	0.611	<	0.755	$1.3 \cdot 10^7$	>	$7.7 \cdot 10^6$	$4.7 \cdot 10^3$	>	$1.5 \cdot 10^3$
housing	0.406	=	0.472	$5.0 \cdot 10^6$	=	$5.4 \cdot 10^6$	$2.4 \cdot 10^3$	>	$1.2 \cdot 10^3$
kin8nm	0.579	<	0.667	$7.8 \cdot 10^6$	<	$2.5 \cdot 10^7$	$1.0 \cdot 10^3$	<	$1.8 \cdot 10^3$
lowbwt	0.601	=	0.602	$2.8 \cdot 10^6$	=	$2.4 \cdot 10^6$	$1.7 \cdot 10^3$	>	$1.1 \cdot 10^3$
mbagrade	0.938	=	0.927	$5.5 \cdot 10^2$	>	$3.5 \cdot 10^2$	$4.4 \cdot 10^1$	>	$2.4 \cdot 10^1$
meta	1.500	=	1.001	$2.2 \cdot 10^7$	=	$1.4 \cdot 10^7$	$2.5 \cdot 10^3$	>	$6.4 \cdot 10^2$
mv	0.026	=	0.025	$7.5 \cdot 10^6$	<	$1.7 \cdot 10^7$	$1.7 \cdot 10^3$	>	$1.0 \cdot 10^3$
pharynx	0.729	=	0.717	$2.8 \cdot 10^6$	<	$4.0 \cdot 10^6$	$8.6 \cdot 10^2$	=	$1.0 \cdot 10^3$
puma32h	0.271	<	0.573	$1.8 \cdot 10^7$	<	$5.1 \cdot 10^7$	$1.8 \cdot 10^3$	=	$1.2 \cdot 10^3$
pw-linear	0.483	>	0.355	$1.2 \cdot 10^6$	<	$4.4 \cdot 10^6$	$5.6 \cdot 10^2$	=	$6.8 \cdot 10^2$
pyrim	0.728	=	0.731	$2.2 \cdot 10^6$	<	$4.0 \cdot 10^6$	$1.0 \cdot 10^3$	=	$7.8 \cdot 10^2$
quake	0.997	=	0.997	$6.8 \cdot 10^5$	=	$1.1 \cdot 10^6$	$1.7 \cdot 10^3$	=	$1.0 \cdot 10^3$
sensory	0.868	=	0.859	$5.6 \cdot 10^6$	<	$8.7 \cdot 10^6$	$10.0 \cdot 10^2$	=	$1.1 \cdot 10^3$
servo	0.483	>	0.402	$1.0 \cdot 10^6$	<	$1.8 \cdot 10^6$	$4.3 \cdot 10^2$	<	$5.7 \cdot 10^2$
strike	0.914	=	0.907	$4.0 \cdot 10^6$	=	$4.1 \cdot 10^6$	$1.2 \cdot 10^3$	>	$9.1 \cdot 10^2$
triazines	1.152	=	0.897	$7.7 \cdot 10^6$	<	$2.6 \cdot 10^7$	$1.4 \cdot 10^3$	>	$1.1 \cdot 10^3$
veteran	0.901	=	0.900	$3.5 \cdot 10^5$	=	$2.3 \cdot 10^5$	$5.1 \cdot 10^2$	=	$4.0 \cdot 10^2$
vineyard	0.632	=	0.667	$3.8 \cdot 10^4$	>	$8.1 \cdot 10^2$	$4.3 \cdot 10^2$	>	$3.9 \cdot 10^1$

Table 58: The performance of polynomial equations learned by MDLCIPER (P) compared to the performance of piecewise linear equations learned by MDLCIPERX with a maximum non-binary degree variables of 2 (Qx).

	rrmse			*ssc*			*mcl*		
dataset	P	t-test	Qx	P	t-test	Qx	P	t-test	Qx
2dplanes	0.227	=	0.227	$8.9 \cdot 10^5$	<	$1.7 \cdot 10^6$	$2.3 \cdot 10^2$	>	$1.8 \cdot 10^2$
auto-price	0.396	=	0.430	$1.7 \cdot 10^5$	>	$3.9 \cdot 10^4$	$5.8 \cdot 10^1$	=	$4.8 \cdot 10^1$
bank32nh	0.655	=	0.666	$4.0 \cdot 10^7$	=	$2.5 \cdot 10^7$	$1.1 \cdot 10^3$	>	$5.1 \cdot 10^2$
baskball	0.774	=	0.774	$2.6 \cdot 10^3$	=	$1.1 \cdot 10^3$	$2.0 \cdot 10^1$	=	$2.0 \cdot 10^1$
bodyfat	0.170	=	0.159	$1.5 \cdot 10^5$	>	$2.0 \cdot 10^4$	$4.7 \cdot 10^1$	=	$1.0 \cdot 10^1$
cal-housing	0.829	=	0.663	$3.1 \cdot 10^6$	>	$9.8 \cdot 10^5$	$1.2 \cdot 10^3$	>	$6.6 \cdot 10^2$
cholesterol	1.008	=	1.008	$5.0 \cdot 10^4$	>	$6.0 \cdot 10^2$	$1.0 \cdot 10^0$	=	$1.0 \cdot 10^0$
cloud	0.437	=	0.375	$9.6 \cdot 10^4$	>	$3.7 \cdot 10^4$	$4.5 \cdot 10^1$	=	$3.6 \cdot 10^1$
cpu-small	0.174	=	1.473	$5.7 \cdot 10^6$	>	$2.9 \cdot 10^6$	$8.8 \cdot 10^2$	>	$5.8 \cdot 10^2$
delta-ailerons	0.569	>	0.558	$1.5 \cdot 10^4$	=	$1.3 \cdot 10^4$	$3.0 \cdot 10^1$	<	$1.2 \cdot 10^2$
delta-elevators	0.631	=	0.626	$2.2 \cdot 10^3$	<	$7.5 \cdot 10^3$	$2.0 \cdot 10^1$	<	$5.1 \cdot 10^1$
elevators	0.355	>	0.346	$1.4 \cdot 10^6$	>	$5.3 \cdot 10^5$	$2.7 \cdot 10^2$	<	$3.1 \cdot 10^2$
elusage	0.401	=	0.401	$3.1 \cdot 10^2$	<	$5.7 \cdot 10^2$	$3.0 \cdot 10^1$	=	$3.0 \cdot 10^1$
fried-delve	0.201	<	0.205	$9.0 \cdot 10^5$	<	$1.1 \cdot 10^7$	$3.3 \cdot 10^2$	<	$7.1 \cdot 10^2$
fruitfly	1.000	=	1.000	$1.1 \cdot 10^2$	=	$1.1 \cdot 10^2$	$0.0 \cdot 10^0$	=	$0.0 \cdot 10^0$
house-8l	0.620	=	0.835	$2.1 \cdot 10^6$	<	$5.8 \cdot 10^6$	$1.1 \cdot 10^3$	=	$1.1 \cdot 10^3$
housing	0.397	=	0.423	$1.4 \cdot 10^6$	>	$1.1 \cdot 10^6$	$2.9 \cdot 10^2$	>	$2.2 \cdot 10^2$
kin8nm	0.500	<	0.607	$2.4 \cdot 10^6$	=	$4.0 \cdot 10^6$	$1.0 \cdot 10^3$	>	$5.7 \cdot 10^2$
lowbwt	0.622	=	0.626	$4.2 \cdot 10^4$	>	$8.3 \cdot 10^2$	$1.0 \cdot 10^1$	=	$1.1 \cdot 10^1$
mbagrade	1.040	=	1.040	$5.6 \cdot 10^1$	=	$4.6 \cdot 10^1$	$2.0 \cdot 10^0$	=	$2.0 \cdot 10^0$
meta	1.052	=	1.500	$2.4 \cdot 10^7$	=	$2.3 \cdot 10^7$	$6.2 \cdot 10^2$	=	$4.8 \cdot 10^2$
mv	0.029	=	0.027	$6.2 \cdot 10^5$	<	$1.8 \cdot 10^6$	$3.5 \cdot 10^2$	=	$3.0 \cdot 10^2$
pharynx	0.743	=	0.743	$1.6 \cdot 10^5$	=	$1.3 \cdot 10^5$	$3.1 \cdot 10^1$	=	$2.5 \cdot 10^1$
puma32h	0.271	=	0.607	$5.0 \cdot 10^3$	<	$1.1 \cdot 10^7$	$4.0 \cdot 10^1$	=	$1.2 \cdot 10^2$
pw-linear	0.420	=	0.368	$2.3 \cdot 10^5$	<	$4.2 \cdot 10^5$	$1.6 \cdot 10^2$	=	$1.4 \cdot 10^2$
pyrim	0.829	=	1.157	$2.6 \cdot 10^5$	<	$3.5 \cdot 10^5$	$4.0 \cdot 10^1$	=	$3.2 \cdot 10^1$
quake	1.000	=	1.000	$5.0 \cdot 10^3$	>	$1.4 \cdot 10^2$	$0.0 \cdot 10^0$	=	$0.0 \cdot 10^0$
sensory	0.925	=	0.955	$2.0 \cdot 10^5$	>	$1.5 \cdot 10^4$	$4.7 \cdot 10^1$	>	$2.0 \cdot 10^1$
servo	0.302	=	0.309	$6.1 \cdot 10^5$	>	$5.3 \cdot 10^5$	$3.0 \cdot 10^2$	=	$3.0 \cdot 10^2$
strike	0.936	=	0.931	$5.1 \cdot 10^5$	>	$4.0 \cdot 10^5$	$8.5 \cdot 10^1$	=	$7.8 \cdot 10^1$
triazines	0.999	=	0.999	$2.0 \cdot 10^3$	<	$4.0 \cdot 10^3$	$2.0 \cdot 10^0$	=	$2.0 \cdot 10^0$
veteran	1.476	=	0.950	$7.3 \cdot 10^4$	>	$2.1 \cdot 10^3$	$5.8 \cdot 10^1$	=	$1.4 \cdot 10^1$
vineyard	0.701	=	0.770	$8.2 \cdot 10^3$	>	$5.3 \cdot 10^2$	$3.5 \cdot 10^1$	=	$3.0 \cdot 10^1$

Table 59: The performance of polynomial equations learned by CVCIPER (P) compared to the performance of piecewise linear equations learned by CVCIPERX with a maximum non-binary degree variables of 2 (Qx).

	rrmse			*ssc*			*mcl*		
dataset	P	t-test	Qx	P	t-test	Qx	P	t-test	Qx
2dplanes	0.228	=	0.227	$1.4 \cdot 10^7$	=	$1.8 \cdot 10^7$	$2.6 \cdot 10^3$	>	$1.5 \cdot 10^3$
auto-price	0.372	=	0.380	$2.8 \cdot 10^6$	>	$1.1 \cdot 10^6$	$2.2 \cdot 10^3$	>	$1.0 \cdot 10^3$
bank32nh	0.670	=	0.672	$5.4 \cdot 10^7$	<	$1.1 \cdot 10^8$	$3.0 \cdot 10^3$	>	$1.8 \cdot 10^3$
baskball	0.829	=	0.809	$2.7 \cdot 10^4$	=	$1.3 \cdot 10^4$	$2.6 \cdot 10^2$	=	$2.3 \cdot 10^2$
bodyfat	0.166	=	0.167	$7.8 \cdot 10^5$	>	$1.5 \cdot 10^5$	$5.8 \cdot 10^2$	=	$4.2 \cdot 10^2$
cal-housing	0.551	=	0.567	$1.3 \cdot 10^7$	>	$4.4 \cdot 10^6$	$4.8 \cdot 10^3$	>	$1.6 \cdot 10^3$
cholesterol	1.025	=	0.977	$5.2 \cdot 10^6$	=	$5.0 \cdot 10^6$	$2.4 \cdot 10^3$	>	$1.2 \cdot 10^3$
cloud	0.507	=	0.406	$8.6 \cdot 10^5$	<	$1.1 \cdot 10^6$	$1.1 \cdot 10^3$	>	$9.2 \cdot 10^2$
cpu-small	0.189	>	0.170	$1.0 \cdot 10^7$	=	$9.0 \cdot 10^6$	$1.3 \cdot 10^3$	<	$2.0 \cdot 10^3$
delta-ailerons	0.550	=	0.553	$4.3 \cdot 10^6$	>	$2.5 \cdot 10^5$	$3.0 \cdot 10^3$	>	$6.5 \cdot 10^2$
delta-elevators	0.603	<	0.610	$3.3 \cdot 10^6$	>	$7.0 \cdot 10^5$	$1.7 \cdot 10^3$	>	$7.1 \cdot 10^2$
elevators	0.367	=	0.366	$9.5 \cdot 10^6$	>	$2.2 \cdot 10^6$	$1.3 \cdot 10^3$	>	$9.0 \cdot 10^2$
elusage	0.453	=	0.425	$1.4 \cdot 10^3$	<	$1.7 \cdot 10^4$	$5.7 \cdot 10^1$	<	$2.4 \cdot 10^2$
fried-delve	0.201	<	0.208	$1.0 \cdot 10^7$	<	$6.4 \cdot 10^7$	$2.5 \cdot 10^3$	=	$2.4 \cdot 10^3$
fruitfly	1.010	=	1.000	$5.8 \cdot 10^3$	=	$1.0 \cdot 10^3$	$2.8 \cdot 10^1$	=	$0.0 \cdot 10^0$
house-8l	0.611	<	0.643	$1.3 \cdot 10^7$	=	$1.3 \cdot 10^7$	$4.7 \cdot 10^3$	>	$1.9 \cdot 10^3$
housing	0.406	=	0.423	$5.0 \cdot 10^6$	<	$6.8 \cdot 10^6$	$2.4 \cdot 10^3$	>	$1.4 \cdot 10^3$
kin8nm	0.579	<	0.621	$7.8 \cdot 10^6$	<	$2.7 \cdot 10^7$	$1.0 \cdot 10^3$	<	$1.4 \cdot 10^3$
lowbwt	0.601	=	0.593	$2.8 \cdot 10^6$	=	$2.7 \cdot 10^6$	$1.7 \cdot 10^3$	=	$1.4 \cdot 10^3$
mbagrade	0.938	=	0.925	$5.5 \cdot 10^2$	=	$6.4 \cdot 10^2$	$4.4 \cdot 10^1$	=	$5.0 \cdot 10^1$
meta	1.500	=	0.983	$2.2 \cdot 10^7$	=	$1.4 \cdot 10^7$	$2.5 \cdot 10^3$	>	$7.5 \cdot 10^2$
mv	0.026	>	0.022	$7.5 \cdot 10^6$	<	$1.7 \cdot 10^7$	$1.7 \cdot 10^3$	>	$1.2 \cdot 10^3$
pharynx	0.729	=	0.716	$2.8 \cdot 10^6$	<	$4.3 \cdot 10^6$	$8.6 \cdot 10^2$	<	$1.1 \cdot 10^3$
puma32h	0.271	<	0.459	$1.8 \cdot 10^7$	<	$7.7 \cdot 10^7$	$1.8 \cdot 10^3$	=	$1.5 \cdot 10^3$
pw-linear	0.483	>	0.350	$1.2 \cdot 10^6$	<	$5.0 \cdot 10^6$	$5.6 \cdot 10^2$	=	$7.5 \cdot 10^2$
pyrim	0.728	=	1.132	$2.2 \cdot 10^6$	<	$4.8 \cdot 10^6$	$1.0 \cdot 10^3$	=	$9.5 \cdot 10^2$
quake	0.997	=	0.994	$6.8 \cdot 10^5$	=	$1.7 \cdot 10^6$	$1.7 \cdot 10^3$	=	$1.7 \cdot 10^3$
sensory	0.868	=	0.859	$5.6 \cdot 10^6$	<	$8.7 \cdot 10^6$	$10.0 \cdot 10^2$	=	$1.1 \cdot 10^3$
servo	0.483	>	0.402	$1.0 \cdot 10^6$	<	$1.8 \cdot 10^6$	$4.3 \cdot 10^2$	<	$5.7 \cdot 10^2$
strike	0.914	=	0.917	$4.0 \cdot 10^6$	<	$5.6 \cdot 10^6$	$1.2 \cdot 10^3$	=	$1.1 \cdot 10^3$
triazines	1.152	=	0.888	$7.7 \cdot 10^6$	<	$2.8 \cdot 10^7$	$1.4 \cdot 10^3$	=	$1.2 \cdot 10^3$
veteran	0.901	=	0.890	$3.5 \cdot 10^5$	=	$4.9 \cdot 10^5$	$5.1 \cdot 10^2$	=	$6.2 \cdot 10^2$
vineyard	0.632	=	0.638	$3.8 \cdot 10^4$	>	$5.2 \cdot 10^3$	$4.3 \cdot 10^2$	>	$1.5 \cdot 10^2$

Table 60: Comparison of polynomial equations learned by MDLCIPER (P) to piecewise polynomial equations learned by MDLCIPERX (Px).

	rrmse			*ssc*			*mcl*		
dataset	P	t-test	Px	P	t-test	Px	P	t-test	Px
2dplanes	0.227	=	0.227	$8.9 \cdot 10^5$	<	$2.1 \cdot 10^6$	$2.3 \cdot 10^2$	=	$2.1 \cdot 10^2$
auto-price	0.396	=	0.396	$1.7 \cdot 10^5$	<	$1.8 \cdot 10^5$	$5.8 \cdot 10^1$	=	$5.8 \cdot 10^1$
bank32nh	0.655	=	0.659	$4.0 \cdot 10^7$	<	$1.1 \cdot 10^8$	$1.1 \cdot 10^3$	=	$1.1 \cdot 10^3$
baskball	0.774	=	0.774	$2.6 \cdot 10^3$	=	$2.8 \cdot 10^3$	$2.0 \cdot 10^1$	=	$2.0 \cdot 10^1$
bodyfat	0.170	=	0.170	$1.5 \cdot 10^5$	=	$1.5 \cdot 10^5$	$4.7 \cdot 10^1$	=	$4.7 \cdot 10^1$
cal-housing	0.829	=	0.552	$3.1 \cdot 10^6$	=	$2.1 \cdot 10^6$	$1.2 \cdot 10^3$	>	$9.2 \cdot 10^2$
cholesterol	1.008	=	1.008	$5.0 \cdot 10^4$	<	$6.2 \cdot 10^4$	$1.0 \cdot 10^0$	=	$1.0 \cdot 10^0$
cloud	0.437	=	0.438	$9.6 \cdot 10^4$	<	$1.2 \cdot 10^5$	$4.5 \cdot 10^1$	=	$4.4 \cdot 10^1$
cpu-small	0.174	=	0.171	$5.7 \cdot 10^6$	>	$2.5 \cdot 10^6$	$8.8 \cdot 10^2$	>	$4.1 \cdot 10^2$
delta-ailerons	0.569	=	0.569	$1.5 \cdot 10^4$	=	$1.7 \cdot 10^4$	$3.0 \cdot 10^1$	=	$3.0 \cdot 10^1$
delta-elevators	0.631	=	0.631	$2.2 \cdot 10^3$	=	$3.0 \cdot 10^3$	$2.0 \cdot 10^1$	=	$2.0 \cdot 10^1$
elevators	0.355	<	0.368	$1.4 \cdot 10^6$	>	$7.8 \cdot 10^5$	$2.7 \cdot 10^2$	>	$1.7 \cdot 10^2$
elusage	0.401	=	0.401	$3.1 \cdot 10^2$	<	$4.2 \cdot 10^2$	$3.0 \cdot 10^1$	=	$3.0 \cdot 10^1$
fried-delve	0.201	=	0.210	$9.0 \cdot 10^5$	<	$2.2 \cdot 10^6$	$3.3 \cdot 10^2$	=	$3.2 \cdot 10^2$
fruitfly	1.000	=	1.000	$1.1 \cdot 10^2$	=	$1.1 \cdot 10^2$	$0.0 \cdot 10^0$	=	$0.0 \cdot 10^0$
house-8l	0.620	=	0.619	$2.1 \cdot 10^6$	<	$2.3 \cdot 10^6$	$1.1 \cdot 10^3$	>	$6.0 \cdot 10^2$
housing	0.397	=	0.407	$1.4 \cdot 10^6$	<	$2.4 \cdot 10^6$	$2.9 \cdot 10^2$	=	$3.0 \cdot 10^2$
kin8nm	0.500	<	0.610	$2.4 \cdot 10^6$	=	$2.2 \cdot 10^6$	$1.0 \cdot 10^3$	>	$4.1 \cdot 10^2$
lowbwt	0.622	=	0.622	$4.2 \cdot 10^4$	>	$4.2 \cdot 10^4$	$1.0 \cdot 10^1$	=	$1.0 \cdot 10^1$
mbagrade	1.040	=	1.040	$5.6 \cdot 10^1$	=	$5.6 \cdot 10^1$	$2.0 \cdot 10^0$	=	$2.0 \cdot 10^0$
meta	1.052	=	1.465	$2.4 \cdot 10^7$	>	$6.1 \cdot 10^6$	$6.2 \cdot 10^2$	>	$1.9 \cdot 10^2$
mv	0.029	=	0.029	$6.2 \cdot 10^5$	<	$1.7 \cdot 10^6$	$3.5 \cdot 10^2$	>	$3.0 \cdot 10^2$
pharynx	0.743	=	0.743	$1.6 \cdot 10^5$	=	$1.6 \cdot 10^5$	$3.1 \cdot 10^1$	=	$3.1 \cdot 10^1$
puma32h	0.271	>	0.258	$5.0 \cdot 10^3$	<	$2.3 \cdot 10^6$	$4.0 \cdot 10^1$	<	$1.0 \cdot 10^2$
pw-linear	0.420	=	0.389	$2.3 \cdot 10^5$	<	$7.1 \cdot 10^5$	$1.6 \cdot 10^2$	=	$1.5 \cdot 10^2$
pyrim	0.829	=	0.893	$2.6 \cdot 10^5$	<	$4.6 \cdot 10^5$	$4.0 \cdot 10^1$	=	$4.3 \cdot 10^1$
quake	1.000	=	1.000	$5.0 \cdot 10^3$	<	$1.3 \cdot 10^4$	$0.0 \cdot 10^0$	=	$0.0 \cdot 10^0$
sensory	0.925	=	0.925	$2.0 \cdot 10^5$	=	$2.0 \cdot 10^5$	$4.7 \cdot 10^1$	=	$4.7 \cdot 10^1$
servo	0.302	=	0.302	$6.1 \cdot 10^5$	=	$6.1 \cdot 10^5$	$3.0 \cdot 10^2$	=	$3.0 \cdot 10^2$
strike	0.936	=	0.974	$5.1 \cdot 10^5$	<	$5.9 \cdot 10^5$	$8.5 \cdot 10^1$	=	$8.6 \cdot 10^1$
triazines	0.999	=	0.999	$2.0 \cdot 10^3$	=	$2.9 \cdot 10^3$	$2.0 \cdot 10^0$	=	$2.0 \cdot 10^0$
veteran	1.476	=	1.270	$7.3 \cdot 10^4$	=	$6.5 \cdot 10^4$	$5.8 \cdot 10^1$	=	$4.9 \cdot 10^1$
vineyard	0.701	=	0.701	$8.2 \cdot 10^3$	>	$6.4 \cdot 10^3$	$3.5 \cdot 10^1$	=	$3.5 \cdot 10^1$

Table 61: Comparison of polynomial equations learned by CVCIPER (P) to piecewise polynomial equations learned by CVCIPERX (Px).

	rrmse			*ssc*			*mcl*		
dataset	P	t-test	Px	P	t-test	Px	P	t-test	Px
2dplanes	0.228	=	0.227	$1.4 \cdot 10^7$	<	$2.1 \cdot 10^7$	$2.6 \cdot 10^3$	>	$2.1 \cdot 10^3$
auto-price	0.372	=	0.376	$2.8 \cdot 10^6$	=	$2.6 \cdot 10^6$	$2.2 \cdot 10^3$	>	$1.9 \cdot 10^3$
bank32nh	0.670	<	0.679	$5.4 \cdot 10^7$	>	$3.4 \cdot 10^7$	$3.0 \cdot 10^3$	>	$1.1 \cdot 10^3$
baskball	0.829	=	0.829	$2.7 \cdot 10^4$	=	$3.0 \cdot 10^4$	$2.6 \cdot 10^2$	=	$2.7 \cdot 10^2$
bodyfat	0.166	=	0.168	$7.8 \cdot 10^5$	=	$7.5 \cdot 10^5$	$5.8 \cdot 10^2$	=	$5.6 \cdot 10^2$
cal-housing	0.551	=	0.547	$1.3 \cdot 10^7$	=	$1.3 \cdot 10^7$	$4.8 \cdot 10^3$	>	$4.0 \cdot 10^3$
cholesterol	1.025	=	1.018	$5.2 \cdot 10^6$	<	$6.6 \cdot 10^6$	$2.4 \cdot 10^3$	=	$2.2 \cdot 10^3$
cloud	0.507	=	0.642	$8.6 \cdot 10^5$	<	$1.1 \cdot 10^6$	$1.1 \cdot 10^3$	=	$1.0 \cdot 10^3$
cpu-small	0.189	>	0.172	$1.0 \cdot 10^7$	<	$1.5 \cdot 10^7$	$1.3 \cdot 10^3$	<	$2.1 \cdot 10^3$
delta-ailerons	0.550	>	0.543	$4.3 \cdot 10^6$	=	$5.0 \cdot 10^6$	$3.0 \cdot 10^3$	=	$3.1 \cdot 10^3$
delta-elevators	0.603	=	0.602	$3.3 \cdot 10^6$	<	$5.0 \cdot 10^6$	$1.7 \cdot 10^3$	=	$1.8 \cdot 10^3$
elevators	0.367	=	0.367	$9.5 \cdot 10^6$	>	$8.9 \cdot 10^6$	$1.3 \cdot 10^3$	=	$1.3 \cdot 10^3$
elusage	0.453	=	0.452	$1.4 \cdot 10^3$	=	$3.7 \cdot 10^3$	$5.7 \cdot 10^1$	=	$8.2 \cdot 10^1$
fried-delve	0.201	=	0.202	$1.0 \cdot 10^7$	<	$2.7 \cdot 10^7$	$2.5 \cdot 10^3$	=	$2.2 \cdot 10^3$
fruitfly	1.010	=	1.010	$5.8 \cdot 10^3$	=	$4.9 \cdot 10^3$	$2.8 \cdot 10^1$	=	$2.8 \cdot 10^1$
house-8l	0.611	=	0.612	$1.3 \cdot 10^7$	<	$2.1 \cdot 10^7$	$4.7 \cdot 10^3$	>	$3.3 \cdot 10^3$
housing	0.406	=	0.430	$5.0 \cdot 10^6$	<	$7.3 \cdot 10^6$	$2.4 \cdot 10^3$	>	$2.1 \cdot 10^3$
kin8nm	0.579	=	0.581	$7.8 \cdot 10^6$	<	$2.4 \cdot 10^7$	$1.0 \cdot 10^3$	<	$1.4 \cdot 10^3$
lowbwt	0.601	=	0.599	$2.8 \cdot 10^6$	=	$2.8 \cdot 10^6$	$1.7 \cdot 10^3$	=	$1.6 \cdot 10^3$
mbagrade	0.938	=	0.938	$5.5 \cdot 10^2$	=	$5.5 \cdot 10^2$	$4.4 \cdot 10^1$	=	$4.4 \cdot 10^1$
meta	1.500	=	1.500	$2.2 \cdot 10^7$	<	$3.3 \cdot 10^7$	$2.5 \cdot 10^3$	=	$2.1 \cdot 10^3$
mv	0.026	>	0.023	$7.5 \cdot 10^6$	<	$1.5 \cdot 10^7$	$1.7 \cdot 10^3$	>	$1.4 \cdot 10^3$
pharynx	0.729	=	0.730	$2.8 \cdot 10^6$	=	$2.8 \cdot 10^6$	$8.6 \cdot 10^2$	=	$8.6 \cdot 10^2$
puma32h	0.271	>	0.257	$1.8 \cdot 10^7$	=	$2.4 \cdot 10^7$	$1.8 \cdot 10^3$	=	$8.0 \cdot 10^2$
pw-linear	0.483	=	0.459	$1.2 \cdot 10^6$	<	$3.2 \cdot 10^6$	$5.6 \cdot 10^2$	=	$7.2 \cdot 10^2$
pyrim	0.728	=	1.500	$2.2 \cdot 10^6$	<	$3.7 \cdot 10^6$	$1.0 \cdot 10^3$	=	$9.3 \cdot 10^2$
quake	0.997	=	0.995	$6.8 \cdot 10^5$	<	$6.1 \cdot 10^6$	$1.7 \cdot 10^3$	<	$4.0 \cdot 10^3$
sensory	0.868	=	0.868	$5.6 \cdot 10^6$	=	$5.6 \cdot 10^6$	$10.0 \cdot 10^2$	=	$10.0 \cdot 10^2$
servo	0.483	=	0.483	$1.0 \cdot 10^6$	=	$1.0 \cdot 10^6$	$4.3 \cdot 10^2$	=	$4.3 \cdot 10^2$
strike	0.914	=	0.937	$4.0 \cdot 10^6$	=	$4.8 \cdot 10^6$	$1.2 \cdot 10^3$	=	$1.3 \cdot 10^3$
triazines	1.152	=	0.891	$7.7 \cdot 10^6$	<	$1.8 \cdot 10^7$	$1.4 \cdot 10^3$	=	$1.3 \cdot 10^3$
veteran	0.901	=	0.911	$3.5 \cdot 10^5$	=	$3.7 \cdot 10^5$	$5.1 \cdot 10^2$	=	$5.0 \cdot 10^2$
vineyard	0.632	=	0.625	$3.8 \cdot 10^4$	=	$3.0 \cdot 10^4$	$4.3 \cdot 10^2$	=	$3.7 \cdot 10^2$

Table 62: Comparison of piecewise polynomial equations learned by CVCIPERX (CV) and MDL-CIPERX (MDL) on the regression task.

	rrmse			*ssc*			*mcl*		
dataset	CV	t-test	MDL	CV	t-test	MDL	CV	t-test	MDL
2dplanes	0.227	=	0.227	$2.1 \cdot 10^7$	>	$2.1 \cdot 10^6$	$2.1 \cdot 10^3$	>	$2.1 \cdot 10^2$
auto-price	0.376	=	0.396	$2.6 \cdot 10^6$	>	$1.8 \cdot 10^5$	$1.9 \cdot 10^3$	>	$5.8 \cdot 10^1$
bank32nh	0.679	>	0.659	$3.4 \cdot 10^7$	<	$1.1 \cdot 10^8$	$1.1 \cdot 10^3$	=	$1.1 \cdot 10^3$
baskball	0.829	=	0.774	$3.0 \cdot 10^4$	>	$2.8 \cdot 10^3$	$2.7 \cdot 10^2$	>	$2.0 \cdot 10^1$
bodyfat	0.168	=	0.170	$7.5 \cdot 10^5$	>	$1.5 \cdot 10^5$	$5.6 \cdot 10^2$	>	$4.7 \cdot 10^1$
cal-housing	0.547	=	0.552	$1.3 \cdot 10^7$	>	$2.1 \cdot 10^6$	$4.0 \cdot 10^3$	>	$9.2 \cdot 10^2$
cholesterol	1.018	=	1.008	$6.6 \cdot 10^6$	>	$6.2 \cdot 10^4$	$2.2 \cdot 10^3$	>	$1.0 \cdot 10^0$
cloud	0.642	=	0.438	$1.1 \cdot 10^6$	>	$1.2 \cdot 10^5$	$1.0 \cdot 10^3$	>	$4.4 \cdot 10^1$
cpu-small	0.172	=	0.171	$1.5 \cdot 10^7$	>	$2.5 \cdot 10^6$	$2.1 \cdot 10^3$	>	$4.1 \cdot 10^2$
delta-ailerons	0.543	<	0.569	$5.0 \cdot 10^6$	>	$1.7 \cdot 10^4$	$3.1 \cdot 10^3$	>	$3.0 \cdot 10^1$
delta-elevators	0.602	<	0.631	$5.0 \cdot 10^6$	>	$3.0 \cdot 10^3$	$1.8 \cdot 10^3$	>	$2.0 \cdot 10^1$
elevators	0.367	=	0.368	$8.9 \cdot 10^6$	>	$7.8 \cdot 10^5$	$1.3 \cdot 10^3$	>	$1.7 \cdot 10^2$
elusage	0.452	=	0.401	$3.7 \cdot 10^3$	=	$4.2 \cdot 10^2$	$8.2 \cdot 10^1$	>	$3.0 \cdot 10^1$
fried-delve	0.202	=	0.210	$2.7 \cdot 10^7$	>	$2.2 \cdot 10^6$	$2.2 \cdot 10^3$	>	$3.2 \cdot 10^2$
fruitfly	1.010	=	1.000	$4.9 \cdot 10^3$	=	$1.1 \cdot 10^2$	$2.8 \cdot 10^1$	=	$0.0 \cdot 10^0$
house-8l	0.612	=	0.619	$2.1 \cdot 10^7$	>	$2.3 \cdot 10^6$	$3.3 \cdot 10^3$	>	$6.0 \cdot 10^2$
housing	0.430	=	0.407	$7.3 \cdot 10^6$	>	$2.4 \cdot 10^6$	$2.1 \cdot 10^3$	>	$3.0 \cdot 10^2$
kin8nm	0.581	<	0.610	$2.4 \cdot 10^7$	>	$2.2 \cdot 10^6$	$1.4 \cdot 10^3$	>	$4.1 \cdot 10^2$
lowbwt	0.599	=	0.622	$2.8 \cdot 10^6$	>	$4.2 \cdot 10^4$	$1.6 \cdot 10^3$	>	$1.0 \cdot 10^1$
mbagrade	0.938	=	1.040	$5.5 \cdot 10^2$	>	$5.6 \cdot 10^1$	$4.4 \cdot 10^1$	>	$2.0 \cdot 10^0$
meta	1.500	=	1.465	$3.3 \cdot 10^7$	>	$6.1 \cdot 10^6$	$2.1 \cdot 10^3$	>	$1.9 \cdot 10^2$
mv	0.023	<	0.029	$1.5 \cdot 10^7$	>	$1.7 \cdot 10^6$	$1.4 \cdot 10^3$	>	$3.0 \cdot 10^2$
pharynx	0.730	=	0.743	$2.8 \cdot 10^6$	>	$1.6 \cdot 10^5$	$8.6 \cdot 10^2$	>	$3.1 \cdot 10^1$
puma32h	0.257	=	0.258	$2.4 \cdot 10^7$	>	$2.3 \cdot 10^6$	$8.0 \cdot 10^2$	>	$1.0 \cdot 10^2$
pw-linear	0.459	=	0.389	$3.2 \cdot 10^6$	>	$7.1 \cdot 10^5$	$7.2 \cdot 10^2$	>	$1.5 \cdot 10^2$
pyrim	1.500	=	0.893	$3.7 \cdot 10^6$	>	$4.6 \cdot 10^5$	$9.3 \cdot 10^2$	>	$4.3 \cdot 10^1$
quake	0.995	=	1.000	$6.1 \cdot 10^6$	>	$1.3 \cdot 10^4$	$4.0 \cdot 10^3$	>	$0.0 \cdot 10^0$
sensory	0.868	<	0.925	$5.6 \cdot 10^6$	>	$2.0 \cdot 10^5$	$10.0 \cdot 10^2$	>	$4.7 \cdot 10^1$
servo	0.483	>	0.302	$1.0 \cdot 10^6$	>	$6.1 \cdot 10^5$	$4.3 \cdot 10^2$	>	$3.0 \cdot 10^2$
strike	0.937	=	0.974	$4.8 \cdot 10^6$	>	$5.9 \cdot 10^5$	$1.3 \cdot 10^3$	>	$8.6 \cdot 10^1$
triazines	0.891	<	0.999	$1.8 \cdot 10^7$	>	$2.9 \cdot 10^3$	$1.3 \cdot 10^3$	>	$2.0 \cdot 10^0$
veteran	0.911	=	1.270	$3.7 \cdot 10^5$	>	$6.5 \cdot 10^4$	$5.0 \cdot 10^2$	>	$4.9 \cdot 10^1$
vineyard	0.625	=	0.701	$3.0 \cdot 10^4$	>	$6.4 \cdot 10^3$	$3.7 \cdot 10^2$	>	$3.5 \cdot 10^1$

Table 63: The predictive performance (in terms of *rrmse*) of MDLCIPERX as compared to linear regression (**LR**), regression trees (**RT**), and model trees (**MT**). The algorithms are implemented in the WEKA data mining software [24].

dataset	MDLCIPER	t-test	**LR**	t-test	**RT**	t-test	**MT**
2dplanes	0.227	<	0.543	=	0.227	=	0.227
auto-price	0.396	=	0.484	=	0.564	=	0.379
bank32nh	0.659	=	0.685	<	0.752	=	0.673
baskball	0.774	=	0.790	=	0.882	=	0.790
bodyfat	0.170	=	0.165	<	0.329	=	0.158
cal-housing	0.552	<	0.603	=	0.515	>	0.486
cholesterol	1.008	=	1.001	=	1.007	=	1.023
cloud	0.438	=	0.387	<	0.738	=	0.381
cpu-small	0.171	<	0.537	<	0.223	=	0.173
delta-ailerons	0.569	=	0.568	=	0.577	=	0.544
delta-elevators	0.631	>	0.610	=	0.622	>	0.600
elevators	0.368	<	0.433	<	0.521	>	0.322
elusage	0.401	=	0.472	=	0.657	=	0.413
fried-delve	0.210	<	0.527	<	0.357	<	0.278
fruitfly	1.000	=	1.000	=	1.000	=	1.000
house-8l	0.619	<	0.788	=	0.622	=	0.594
housing	0.407	<	0.533	<	0.523	=	0.407
kin8nm	0.610	<	0.766	<	0.688	=	0.607
lowbwt	0.622	=	0.605	=	0.640	=	0.609
mbagrade	1.040	=	0.891	=	1.000	=	0.891
meta	1.465	=	0.992	=	0.992	=	1.031
mv	0.029	<	0.431	<	0.048	=	0.013
pharynx	0.743	<	1.190	<	1.095	<	1.053
puma32h	0.258	<	0.885	<	0.290	=	0.270
pw-linear	0.389	<	0.505	<	0.569	=	0.324
pyrim	0.893	=	0.932	=	1.023	=	0.695
quake	1.000	=	0.998	=	1.001	=	0.996
sensory	0.925	=	0.925	=	0.911	>	0.861
servo	0.302	<	0.541	<	0.628	=	0.352
strike	0.974	=	0.912	=	0.942	=	0.910
triazines	0.999	=	0.968	>	0.901	=	0.849
veteran	1.270	=	0.903	=	0.950	=	0.897
vineyard	0.701	=	0.665	=	0.832	=	0.674

Table 64: The predictive performance (in terms of *rrmse*) of CVCIPERX as compared to linear regression (**LR**), regression trees (**RT**), and model trees (**MT**). The algorithms are implemented in the WEKA data mining software [24].

dataset	MDLCIPER	t-test	**LR**	t-test	**RT**	t-test	**MT**
2dplanes	0.228	<	0.543	=	0.227	=	0.227
auto-price	0.372	=	0.484	=	0.564	=	0.379
bank32nh	0.670	=	0.685	<	0.752	=	0.673
baskball	0.829	=	0.790	=	0.882	=	0.790
bodyfat	0.166	=	0.165	<	0.329	=	0.158
cal-housing	0.551	<	0.603	>	0.515	>	0.486
cholesterol	1.025	=	1.001	=	1.007	=	1.023
cloud	0.507	=	0.387	<	0.738	=	0.381
cpu-small	0.189	<	0.537	<	0.223	=	0.173
delta-ailerons	0.550	=	0.568	=	0.577	=	0.544
delta-elevators	0.603	=	0.610	=	0.622	=	0.600
elevators	0.367	<	0.433	<	0.521	>	0.322
elusage	0.453	=	0.472	=	0.657	=	0.413
fried-delve	0.201	<	0.527	<	0.357	<	0.278
fruitfly	1.010	=	1.000	=	1.000	=	1.000
house-8l	0.611	<	0.788	=	0.622	=	0.594
housing	0.406	<	0.533	<	0.523	=	0.407
kin8nm	0.579	<	0.766	<	0.688	<	0.607
lowbwt	0.601	=	0.605	=	0.640	=	0.609
mbagrade	0.938	=	0.891	=	1.000	=	0.891
meta	1.500	=	0.992	=	0.992	=	1.031
mv	0.026	<	0.431	<	0.048	=	0.013
pharynx	0.729	<	1.190	<	1.095	<	1.053
puma32h	0.271	<	0.885	<	0.290	=	0.270
pw-linear	0.483	=	0.505	=	0.569	>	0.324
pyrim	0.728	=	0.932	<	1.023	=	0.695
quake	0.997	=	0.998	=	1.001	=	0.996
sensory	0.868	<	0.925	<	0.911	=	0.861
servo	0.483	=	0.541	=	0.628	>	0.352
strike	0.914	=	0.912	=	0.942	=	0.910
triazines	1.152	=	0.968	=	0.901	=	0.849
veteran	0.901	=	0.903	=	0.950	=	0.897
vineyard	0.632	=	0.665	=	0.832	=	0.674

B.2 Multi-target Regression

Table 65: The performance of Ciper on the task of multitarget regression. MDLCiper - Ciper with the MDL heuristic (MDL) and CVCiper - Ciper with the CV heuristic (CV).

target	*rrmse*			*ssc*			*mcl*		
dataset	MDL	t-test	CV	MDL	t-test	CV	MDL	t-test	CV
DFlow	0.82		0.77						
DGap	0.75		0.74						
edm		=		$4.6 \cdot 10^5$	<	$2.2 \cdot 10^6$	$1.2 \cdot 10^2$	<	$1.5 \cdot 10^3$
c-class	1.06		0.99						
m-class	1.04		1.01						
x-class	1.30		1.02						
solar-flare1		=		$3.4 \cdot 10^5$	<	$5.0 \cdot 10^6$	$8.4 \cdot 10^1$	<	$1.4 \cdot 10^3$
c-class	1.07		0.96						
m-class	1.26		1.03						
x-class	1.16		0.95						
solar-flare2		=		$1.9 \cdot 10^6$	<	$1.0 \cdot 10^7$	$2.8 \cdot 10^2$	<	$2.0 \cdot 10^3$
Cladophora	2.87		1.18						
Gongrosira	3.68		1.03						
Oedogonium	4.78		1.12						
Stigeoclonium	4.11		1.31						
Melosira	1.39		1.11						
Nitzschia	2.24		1.19						
Audouinella	2.54		1.10						
Erpobdella	3.99		1.32						
Gammarus	6.16		0.91						
Baetis	1.90		1.03						
Hydropsyche	6.54		0.99						
Rhyacophila	3.55		1.13						
Simulium	5.96		1.17						
Tubifex	1.90		1.05						
water-quality		>		$1.3 \cdot 10^7$	=	$8.7 \cdot 10^6$	$1.2 \cdot 10^3$	<	$2.2 \cdot 10^3$
Large Tree Score	0.79		0.79						
Tree Canopy Score	0.65		0.65						
Understorey Score	0.64		0.63						
Litter Score	0.64		0.64						
Logs Score	0.65		0.65						
Weeds Score	0.56		0.56						
Recruitment Score	0.73		0.73						
habitat-hectares		>		$1.2 \cdot 10^7$	<	$8.4 \cdot 10^7$	$5.1 \cdot 10^2$	<	$3.2 \cdot 10^3$

Continued on next page

Table 65 – continued from previous page

targets	*rrmse*			*ssc*			*mcl*		
dataset	MDL	t-test	CV	MDL	t-test	CV	MDL	t-test	CV
Temp	1.02		0.98						
DisO	1.12		1.01						
SD	1.04		1.01						
Conduc	1.25		1.00						
pH	2.00		1.02						
NO2	1.04		1.02						
NO3	1.02		1.00						
NH4	1.02		1.00						
InorgN	1.00		1.03						
SO4	1.08		1.00						
Na	1.02		1.00						
K	1.09		1.00						
Mg	1.09		1.02						
Cu	1.01		1.00						
Mn	1.06		1.02						
prespa-lake		$>$		$1.0 \cdot 10^7$	$>$	$4.0 \cdot 10^5$	$1.7 \cdot 10^2$	$=$	$1.2 \cdot 10^2$
Heat Resistance	1.09		1.50						
Heat Resilience	1.03		1.62						
Cu Resistance	0.86		0.98						
Cu Resilience	0.94		1.61						
Over resist eg	1.13		1.07						
Over resil 2dc eg	0.77		0.71						
1/Cc	0.81		0.98						
Ce/Cc	0.82		1.07						
soil-resilience		$=$		$4.8 \cdot 10^3$	$<$	$1.3 \cdot 10^5$	$1.0 \cdot 10^1$	$<$	$3.1 \cdot 10^2$
MF0/00 plR	0.91		0.68						
MS0/00 plR	0.92		0.60						
sigmea-real		$=$		$2.8 \cdot 10^5$	$<$	$4.9 \cdot 10^5$	$6.9 \cdot 10^2$	$<$	$1.4 \cdot 10^3$
Dispersal Rate Pollen	0.07		0.09						
Dispersal Seeds	0.05		0.07						
sigmea-simulated		$<$		$4.5 \cdot 10^6$	$<$	$1.1 \cdot 10^7$	$8.1 \cdot 10^2$	$<$	$2.7 \cdot 10^3$
acari	0.74		0.80						
collembola	0.75		0.80						
sh-biodiv	0.78		0.88						
soil-quality		$<$		$7.6 \cdot 10^7$	$>$	$4.0 \cdot 10^7$	$8.5 \cdot 10^2$	$<$	$2.2 \cdot 10^3$

Table 66: The performance of CIPER on the task of piecewise multitarget regression: comparison of MDLCIPERX (MDL) and CVCIPERX (CV) on the multitarget regression task.

target	*rrmse*			*ssc*			*mcl*		
dataset	MDL	t-test	CV	MDL	t-test	CV	MDL	t-test	CV
DFlow	0.77		0.71						
DGap	0.79		0.73						
edm		=		$6.0 \cdot 10^5$	<	$4.1 \cdot 10^6$	$9.8 \cdot 10^1$	<	$1.6 \cdot 10^3$
c-class	1.06		0.99						
m-class	1.04		1.01						
x-class	1.30		1.02						
solar-flare1		=		$3.4 \cdot 10^5$	<	$5.0 \cdot 10^6$	$8.4 \cdot 10^1$	<	$1.4 \cdot 10^3$
c-class	1.07		0.96						
m-class	1.26		1.03						
x-class	1.16		0.95						
solar-flare2		=		$1.9 \cdot 10^6$	<	$1.0 \cdot 10^7$	$2.8 \cdot 10^2$	<	$2.0 \cdot 10^3$
Cladophora	2.88		1.18						
Gongrosira	3.69		1.03						
Oedogonium	4.78		1.12						
Stigeoclonium	4.14		1.31						
Melosira	1.40		1.11						
Nitzschia	2.25		1.19						
Audouinella	2.57		1.10						
Erpobdella	3.99		1.32						
Gammarus	6.17		0.91						
Baetis	1.9		1.03						
Hydropsyche	6.54		0.99						
Rhyacophila	3.55		1.13						
Simulium	5.96		1.17						
Tubifex	1.92		1.05						
water-quality		>		$1.5 \cdot 10^7$	>	$8.7 \cdot 10^6$	$1.3 \cdot 10^3$	<	$2.2 \cdot 10^3$
Large Tree Score	0.79		0.79						
Tree Canopy Score	0.65		0.66						
Understorey Score	0.63		0.64						
Litter Score	0.64		0.65						
Logs Score	0.65		0.66						
Weeds Score	0.56		0.57						
Recruitment Score	0.73		0.73						
habitat-hectares		<		$2.1 \cdot 10^7$	<	$1.0 \cdot 10^8$	$5.3 \cdot 10^2$	<	$2.5 \cdot 10^3$

Continued on next page

Table 66 – continued from previous page

target	*rrmse*			*ssc*			*mcl*		
dataset	MDL	t-test	CV	MDL	t-test	CV	MDL	t-test	CV
Temp	1.02		0.98						
DisO	1.02		1.01						
SD	0.99		1.01						
Conduc	1.03		1.00						
pH	1.01		1.02						
NO2	1.01		1.01						
NO3	0.99		1.00						
NH4	0.99		1.00						
InorgN	1.00		1.03						
SO4	1.01		1.00						
Na	1.02		1.00						
K	0.99		1.00						
Mg	1.06		1.02						
Cu	1.00		1.00						
Mn	1.02		1.02						
prespa-lake		=		$6.1 \cdot 10^6$	=	$6.6 \cdot 10^5$	$5.4 \cdot 10^1$	=	$1.3 \cdot 10^2$
Heat Resistance	1.09		1.50						
Heat Resilience	1.03		1.28						
Cu Resistance	0.86		0.75						
Cu Resilience	0.94		1.50						
Over resist eg	1.13		1.24						
Over resil 2dc eg	0.77		0.86						
1/Cc	0.81		0.93						
Ce/Cc	0.82		0.94						
soil-resilience		=		$4.8 \cdot 10^3$	<	$1.2 \cdot 10^5$	$1.0 \cdot 10^1$	<	$3.0 \cdot 10^2$
MF0/00 plR	1.80		0.75						
MS0/00 plR	0.90		0.66						
sigmea-real		=		$2.9 \cdot 10^6$	<	$5.4 \cdot 10^6$	$1.1 \cdot 10^3$	<	$3.7 \cdot 10^3$
Dispersal Rate Pollen	1.50		0.08						
Dispersal Seeds	1.50		0.06						
sigmea-simulated		=		$7.8 \cdot 10^6$	=	$3.7 \cdot 10^7$	$7.3 \cdot 10^2$	<	$3.3 \cdot 10^3$
acari	0.73		0.80						
collembola	0.76		0.80						
sh-biodiv	0.79		0.85						
soil-quality		<		$9.2 \cdot 10^7$	=	$5.3 \cdot 10^7$	$8.3 \cdot 10^2$	<	$2.1 \cdot 10^3$

B.3 Classification via Multi-Target Regression

Table 67: The performance of MDLCiper - Ciper with the improved MDL heuristic on the task of classification via regression. A comparison of quadratic equations - with a maximum degree of 2 (Q) and linear equations - with a maximum degree of 1 (L).

	ce			*ssc*			*mcl*		
dataset	Q	t-test	L	Q	t-test	L	Q	t-test	L
mushroom	0.000	$<$	0.058	$1.1 \cdot 10^{6}$	$>$	$1.0 \cdot 10^{4}$	$1.1 \cdot 10^{2}$	$>$	$3.0 \cdot 10^{1}$
vote	0.043	$=$	0.043	$2.0 \cdot 10^{4}$	$>$	$1.2 \cdot 10^{3}$	$3.7 \cdot 10^{1}$	$>$	$2.8 \cdot 10^{1}$
glass	0.467	$=$	0.491	$1.0 \cdot 10^{4}$	$>$	$6.2 \cdot 10^{2}$	$1.3 \cdot 10^{2}$	$>$	$4.0 \cdot 10^{1}$
segment	0.159	$<$	0.298	$4.1 \cdot 10^{4}$	$>$	$1.2 \cdot 10^{3}$	$1.6 \cdot 10^{2}$	$>$	$4.0 \cdot 10^{1}$
anneal	0.126	$<$	0.164	$2.2 \cdot 10^{5}$	$>$	$2.6 \cdot 10^{3}$	$1.3 \cdot 10^{2}$	$>$	$2.0 \cdot 10^{1}$
sick	0.063	$=$	0.062	$8.4 \cdot 10^{4}$	$>$	$1.3 \cdot 10^{3}$	$1.1 \cdot 10^{2}$	$>$	$2.0 \cdot 10^{1}$
zoo	0.277	$=$	0.257	$6.4 \cdot 10^{5}$	$>$	$1.0 \cdot 10^{4}$	$6.5 \cdot 10^{1}$	$>$	$2.6 \cdot 10^{1}$
ionosphere	0.103	$<$	0.171	$2.3 \cdot 10^{5}$	$>$	$2.6 \cdot 10^{3}$	$1.6 \cdot 10^{2}$	$>$	$3.0 \cdot 10^{1}$
cleveland-14	0.186	$=$	0.159	$7.9 \cdot 10^{4}$	$>$	$1.3 \cdot 10^{3}$	$8.6 \cdot 10^{1}$	$>$	$3.0 \cdot 10^{1}$
german credit	0.273	$=$	0.274	$5.4 \cdot 10^{5}$	$>$	$5.1 \cdot 10^{3}$	$1.4 \cdot 10^{2}$	$>$	$2.0 \cdot 10^{1}$
lymphography	0.392	$=$	0.432	$1.5 \cdot 10^{5}$	$>$	$2.6 \cdot 10^{3}$	$6.1 \cdot 10^{1}$	$>$	$2.1 \cdot 10^{1}$
hepatitis	0.229	$=$	0.220	$2.6 \cdot 10^{3}$	$>$	$6.6 \cdot 10^{2}$	$1.5 \cdot 10^{1}$	$=$	$1.4 \cdot 10^{1}$
autos	0.446	$=$	0.440	$5.6 \cdot 10^{5}$	$>$	$5.1 \cdot 10^{3}$	$1.2 \cdot 10^{2}$	$>$	$4.1 \cdot 10^{1}$
hungarian-14	0.234	$=$	0.195	$3.9 \cdot 10^{4}$	$>$	$1.2 \cdot 10^{3}$	$7.4 \cdot 10^{1}$	$>$	$3.4 \cdot 10^{1}$
vowel	0.436	$<$	0.641	$1.6 \cdot 10^{5}$	$>$	$1.3 \cdot 10^{3}$	$2.7 \cdot 10^{2}$	$>$	$3.0 \cdot 10^{1}$
heart-statlog	0.178	$=$	0.211	$2.0 \cdot 10^{4}$	$>$	$1.1 \cdot 10^{3}$	$6.3 \cdot 10^{1}$	$>$	$3.1 \cdot 10^{1}$
balance-scale	0.093	$<$	0.138	$1.2 \cdot 10^{3}$	$>$	$1.6 \cdot 10^{2}$	$6.6 \cdot 10^{1}$	$>$	$4.0 \cdot 10^{1}$
vehicle	0.294	$<$	0.404	$4.1 \cdot 10^{4}$	$>$	$1.2 \cdot 10^{3}$	$1.5 \cdot 10^{2}$	$>$	$3.0 \cdot 10^{1}$
pima-indians	0.233	$=$	0.245	$5.1 \cdot 10^{3}$	$=$	$6.0 \cdot 10^{2}$	$7.1 \cdot 10^{1}$	$>$	$3.0 \cdot 10^{1}$
sonar	0.317	$=$	0.298	$3.4 \cdot 10^{5}$	$>$	$4.7 \cdot 10^{3}$	$5.6 \cdot 10^{1}$	$>$	$3.1 \cdot 10^{1}$

Table 68: The performance of CVCiper - Ciper using the CV heuristic on the task of classification via regression. A comparison of quadratic equations - with a maximum degree of 2 (Q) and linear equations - with a maximum degree of 1 (L).

		ce			*ssc*			*mcl*	
dataset	Q	t-test	L	Q	t-test	L	Q	t-test	L
mushroom	0.001	$<$	0.049	$8.4 \cdot 10^6$	$>$	$1.0 \cdot 10^5$	$7.4 \cdot 10^2$	$>$	$1.0 \cdot 10^2$
vote	0.046	$=$	0.043	$1.9 \cdot 10^5$	$>$	$1.1 \cdot 10^4$	$5.9 \cdot 10^2$	$>$	$1.0 \cdot 10^2$
glass	0.533	$=$	0.561	$8.3 \cdot 10^3$	$>$	$1.3 \cdot 10^3$	$1.4 \cdot 10^2$	$>$	$5.2 \cdot 10^1$
segment	0.189	$<$	0.295	$2.0 \cdot 10^5$	$>$	$1.1 \cdot 10^4$	$3.4 \cdot 10^2$	$>$	$6.0 \cdot 10^1$
anneal	0.063	$<$	0.164	$2.4 \cdot 10^6$	$>$	$1.7 \cdot 10^4$	$7.8 \cdot 10^2$	$>$	$8.1 \cdot 10^1$
sick	0.062	$=$	0.062	$7.9 \cdot 10^5$	$>$	$1.3 \cdot 10^4$	$8.1 \cdot 10^2$	$>$	$8.3 \cdot 10^1$
zoo	0.119	$=$	0.238	$2.0 \cdot 10^6$	$>$	$2.9 \cdot 10^4$	$3.0 \cdot 10^2$	$>$	$1.2 \cdot 10^2$
ionosphere	0.063	$<$	0.154	$2.1 \cdot 10^6$	$>$	$2.4 \cdot 10^4$	$7.8 \cdot 10^2$	$>$	$1.1 \cdot 10^2$
cleveland-14	0.149	$=$	0.159	$7.5 \cdot 10^5$	$>$	$1.3 \cdot 10^4$	$7.8 \cdot 10^2$	$>$	$9.1 \cdot 10^1$
german credit	0.244	$<$	0.283	$6.1 \cdot 10^6$	$>$	$4.8 \cdot 10^4$	$1.3 \cdot 10^3$	$>$	$1.3 \cdot 10^2$
lymphography	0.182	$=$	0.196	$1.5 \cdot 10^6$	$>$	$2.2 \cdot 10^4$	$6.8 \cdot 10^2$	$>$	$1.3 \cdot 10^2$
hepatitis	0.212	$=$	0.203	$2.6 \cdot 10^5$	$>$	$7.8 \cdot 10^3$	$5.2 \cdot 10^2$	$>$	$8.3 \cdot 10^1$
autos	0.492	$=$	0.513	$5.7 \cdot 10^5$	$>$	$1.2 \cdot 10^4$	$2.3 \cdot 10^2$	$>$	$9.2 \cdot 10^1$
hungarian-14	0.184	$=$	0.184	$3.6 \cdot 10^5$	$>$	$1.1 \cdot 10^4$	$6.3 \cdot 10^2$	$>$	$1.0 \cdot 10^2$
vowel	0.388	$<$	0.639	$1.6 \cdot 10^6$	$>$	$1.3 \cdot 10^4$	$7.1 \cdot 10^2$	$>$	$4.5 \cdot 10^1$
heart-statlog	0.167	$=$	0.174	$1.9 \cdot 10^5$	$>$	$1.0 \cdot 10^4$	$6.0 \cdot 10^2$	$>$	$9.2 \cdot 10^1$
balance-scale	0.088	$<$	0.134	$1.2 \cdot 10^4$	$>$	$1.6 \cdot 10^3$	$1.7 \cdot 10^2$	$>$	$4.0 \cdot 10^1$
vehicle	0.262	$<$	0.391	$4.1 \cdot 10^5$	$>$	$1.2 \cdot 10^4$	$8.9 \cdot 10^2$	$>$	$7.4 \cdot 10^1$
pima-indians	0.233	$=$	0.233	$5.0 \cdot 10^4$	$>$	$5.3 \cdot 10^3$	$2.9 \cdot 10^2$	$>$	$7.5 \cdot 10^1$
sonar	0.221	$=$	0.236	$5.3 \cdot 10^6$	$>$	$5.1 \cdot 10^4$	$1.1 \cdot 10^3$	$>$	$1.7 \cdot 10^2$

Table 69: The performance of MDLCIPER on the task of classification via regression. A comparison of polynomial equations (P) and quadratic equations - with a maximum degree of 2 (Q).

	ce			*ssc*			*mcl*		
dataset	P	t-test	Q	P	t-test	Q	P	t-test	Q
mushroom	0.000	=	0.000	$1.7 \cdot 10^6$	>	$1.1 \cdot 10^6$	$1.1 \cdot 10^2$	=	$1.1 \cdot 10^2$
vote	0.060	=	0.043	$3.1 \cdot 10^5$	>	$2.0 \cdot 10^4$	$2.1 \cdot 10^2$	>	$3.7 \cdot 10^1$
glass	0.322	<	0.467	$3.8 \cdot 10^6$	>	$1.0 \cdot 10^4$	$1.3 \cdot 10^3$	>	$1.3 \cdot 10^2$
segment	0.080	<	0.159	$2.1 \cdot 10^6$	>	$4.1 \cdot 10^4$	$4.4 \cdot 10^2$	>	$1.6 \cdot 10^2$
anneal	0.124	=	0.126	$8.6 \cdot 10^5$	>	$2.2 \cdot 10^5$	$8.3 \cdot 10^1$	<	$1.3 \cdot 10^2$
sick	0.065	=	0.063	$4.7 \cdot 10^5$	>	$8.4 \cdot 10^4$	$1.6 \cdot 10^2$	>	$1.1 \cdot 10^2$
zoo	0.238	=	0.277	$1.1 \cdot 10^6$	>	$6.4 \cdot 10^5$	$8.0 \cdot 10^1$	=	$6.5 \cdot 10^1$
ionosphere	0.077	=	0.103	$2.4 \cdot 10^6$	>	$2.3 \cdot 10^5$	$1.8 \cdot 10^2$	=	$1.6 \cdot 10^2$
cleveland-14	0.179	=	0.186	$6.5 \cdot 10^5$	>	$7.9 \cdot 10^4$	$9.7 \cdot 10^1$	=	$8.6 \cdot 10^1$
german credit	0.276	=	0.273	$1.3 \cdot 10^6$	>	$5.4 \cdot 10^5$	$1.0 \cdot 10^2$	=	$1.4 \cdot 10^2$
lymphography	0.405	=	0.392	$4.3 \cdot 10^5$	>	$1.5 \cdot 10^5$	$7.9 \cdot 10^1$	=	$6.1 \cdot 10^1$
hepatitis	0.229	=	0.229	$3.6 \cdot 10^3$	=	$2.6 \cdot 10^3$	$1.5 \cdot 10^1$	=	$1.5 \cdot 10^1$
autos	0.430	=	0.446	$1.6 \cdot 10^6$	>	$5.6 \cdot 10^5$	$1.4 \cdot 10^2$	=	$1.2 \cdot 10^2$
hungarian-14	0.241	=	0.234	$2.7 \cdot 10^5$	>	$3.9 \cdot 10^4$	$1.0 \cdot 10^2$	=	$7.4 \cdot 10^1$
vowel	0.397	=	0.436	$1.8 \cdot 10^6$	>	$1.6 \cdot 10^5$	$3.8 \cdot 10^2$	>	$2.7 \cdot 10^2$
heart-statlog	0.185	=	0.178	$2.2 \cdot 10^5$	>	$2.0 \cdot 10^4$	$8.3 \cdot 10^1$	>	$6.3 \cdot 10^1$
balance-scale	0.090	=	0.093	$1.4 \cdot 10^5$	>	$1.2 \cdot 10^3$	$4.5 \cdot 10^2$	>	$6.6 \cdot 10^1$
vehicle	0.225	<	0.294	$2.5 \cdot 10^6$	>	$4.1 \cdot 10^4$	$4.8 \cdot 10^2$	>	$1.5 \cdot 10^2$
pima-indians	0.234	=	0.233	$1.6 \cdot 10^5$	>	$5.1 \cdot 10^3$	$1.3 \cdot 10^2$	>	$7.1 \cdot 10^1$
sonar	0.288	=	0.317	$9.1 \cdot 10^5$	>	$3.4 \cdot 10^5$	$5.0 \cdot 10^1$	=	$5.6 \cdot 10^1$

Table 70: The performance of CVCIPER on the task of classification via regression. A comparison of polynomial equations (P) and quadratic equations - with a maximum degree of 2 (Q).

		ce			*ssc*			*mcl*	
dataset	P	t-test	Q	P	t-test	Q	P	t-test	Q
mushroom	0.000	=	0.001	$1.4 \cdot 10^7$	>	$8.4 \cdot 10^6$	$7.4 \cdot 10^2$	=	$7.4 \cdot 10^2$
vote	0.050	=	0.046	$1.5 \cdot 10^7$	>	$1.9 \cdot 10^5$	$3.2 \cdot 10^3$	>	$5.9 \cdot 10^2$
glass	0.327	<	0.533	$1.2 \cdot 10^8$	>	$8.3 \cdot 10^3$	$1.5 \cdot 10^4$	>	$1.4 \cdot 10^2$
segment	0.060	<	0.189	$4.9 \cdot 10^7$	>	$2.0 \cdot 10^5$	$3.6 \cdot 10^3$	>	$3.4 \cdot 10^2$
anneal	0.018	=	0.063	$5.2 \cdot 10^7$	>	$2.4 \cdot 10^6$	$2.5 \cdot 10^3$	>	$7.8 \cdot 10^2$
sick	0.063	=	0.062	$5.7 \cdot 10^7$	>	$7.9 \cdot 10^5$	$5.1 \cdot 10^3$	>	$8.1 \cdot 10^2$
zoo	0.059	=	0.119	$2.3 \cdot 10^7$	>	$2.0 \cdot 10^6$	$1.2 \cdot 10^3$	>	$3.0 \cdot 10^2$
ionosphere	0.060	=	0.063	$1.2 \cdot 10^8$	>	$2.1 \cdot 10^6$	$5.1 \cdot 10^3$	>	$7.8 \cdot 10^2$
cleveland-14	0.172	=	0.149	$7.8 \cdot 10^7$	>	$7.5 \cdot 10^5$	$6.1 \cdot 10^3$	>	$7.8 \cdot 10^2$
german credit	0.244	=	0.244	$7.5 \cdot 10^7$	>	$6.1 \cdot 10^6$	$2.9 \cdot 10^3$	>	$1.3 \cdot 10^3$
lymphography	0.209	=	0.182	$1.9 \cdot 10^8$	>	$1.5 \cdot 10^6$	$7.0 \cdot 10^3$	>	$6.8 \cdot 10^2$
hepatitis	0.186	=	0.212	$8.8 \cdot 10^7$	>	$2.6 \cdot 10^5$	$8.3 \cdot 10^3$	>	$5.2 \cdot 10^2$
autos	0.212	<	0.492	$1.2 \cdot 10^8$	>	$5.7 \cdot 10^5$	$4.2 \cdot 10^3$	>	$2.3 \cdot 10^2$
hungarian-14	0.215	=	0.184	$5.6 \cdot 10^7$	>	$3.6 \cdot 10^5$	$6.3 \cdot 10^3$	>	$6.3 \cdot 10^2$
vowel	0.298	<	0.388	$3.5 \cdot 10^7$	>	$1.6 \cdot 10^6$	$2.2 \cdot 10^3$	>	$7.1 \cdot 10^2$
heart-statlog	0.193	=	0.167	$6.5 \cdot 10^7$	>	$1.9 \cdot 10^5$	$7.6 \cdot 10^3$	>	$6.0 \cdot 10^2$
balance-scale	0.085	=	0.088	$2.4 \cdot 10^7$	>	$1.2 \cdot 10^4$	$8.2 \cdot 10^3$	>	$1.7 \cdot 10^2$
vehicle	0.190	<	0.262	$5.4 \cdot 10^7$	>	$4.1 \cdot 10^5$	$5.2 \cdot 10^3$	>	$8.9 \cdot 10^2$
pima-indians	0.228	=	0.233	$3.3 \cdot 10^7$	>	$5.0 \cdot 10^4$	$7.0 \cdot 10^3$	>	$2.9 \cdot 10^2$
sonar	0.144	=	0.221	$1.6 \cdot 10^8$	>	$5.3 \cdot 10^6$	$5.0 \cdot 10^3$	>	$1.1 \cdot 10^3$

Table 71: Comparison of the performance of CVCIPER (CV) vs MDLCIPER (MDL) on the task of classification via regression.

		ce			*ssc*			*mcl*	
dataset	CV	t-test	MDL	CV	t-test	MDL	CV	t-test	MDL
mushroom	0.000	=	0.000	$1.4 \cdot 10^7$	>	$1.7 \cdot 10^6$	$7.4 \cdot 10^2$	>	$1.1 \cdot 10^2$
vote	0.050	=	0.060	$1.5 \cdot 10^7$	>	$3.1 \cdot 10^5$	$3.2 \cdot 10^3$	>	$2.1 \cdot 10^2$
glass	0.327	=	0.322	$1.2 \cdot 10^8$	>	$3.8 \cdot 10^6$	$1.5 \cdot 10^4$	>	$1.3 \cdot 10^3$
segment	0.060	<	0.080	$4.9 \cdot 10^7$	>	$2.1 \cdot 10^6$	$3.6 \cdot 10^3$	>	$4.4 \cdot 10^2$
anneal	0.018	<	0.124	$5.2 \cdot 10^7$	>	$8.6 \cdot 10^5$	$2.5 \cdot 10^3$	>	$8.3 \cdot 10^1$
sick	0.063	=	0.065	$5.7 \cdot 10^7$	>	$4.7 \cdot 10^5$	$5.1 \cdot 10^3$	>	$1.6 \cdot 10^2$
zoo	0.059	=	0.238	$2.3 \cdot 10^7$	>	$1.1 \cdot 10^6$	$1.2 \cdot 10^3$	>	$8.0 \cdot 10^1$
ionosphere	0.060	=	0.077	$1.2 \cdot 10^8$	>	$2.4 \cdot 10^6$	$5.1 \cdot 10^3$	>	$1.8 \cdot 10^2$
cleveland-14	0.172	=	0.179	$7.8 \cdot 10^7$	>	$6.5 \cdot 10^5$	$6.1 \cdot 10^3$	>	$9.7 \cdot 10^1$
german credit	0.244	<	0.276	$7.5 \cdot 10^7$	>	$1.3 \cdot 10^6$	$2.9 \cdot 10^3$	>	$1.0 \cdot 10^2$
lymphography	0.209	<	0.405	$1.9 \cdot 10^8$	>	$4.3 \cdot 10^5$	$7.0 \cdot 10^3$	>	$7.9 \cdot 10^1$
hepatitis	0.186	=	0.229	$8.8 \cdot 10^7$	>	$3.6 \cdot 10^3$	$8.3 \cdot 10^3$	>	$1.5 \cdot 10^1$
autos	0.212	<	0.430	$1.2 \cdot 10^8$	>	$1.6 \cdot 10^6$	$4.2 \cdot 10^3$	>	$1.4 \cdot 10^2$
hungarian-14	0.215	=	0.241	$5.6 \cdot 10^7$	>	$2.7 \cdot 10^5$	$6.3 \cdot 10^3$	>	$1.0 \cdot 10^2$
vowel	0.298	<	0.397	$3.5 \cdot 10^7$	>	$1.8 \cdot 10^6$	$2.2 \cdot 10^3$	>	$3.8 \cdot 10^2$
heart-statlog	0.193	=	0.185	$6.5 \cdot 10^7$	>	$2.2 \cdot 10^5$	$7.6 \cdot 10^3$	>	$8.3 \cdot 10^1$
balance-scale	0.085	=	0.090	$2.4 \cdot 10^7$	>	$1.4 \cdot 10^5$	$8.2 \cdot 10^3$	>	$4.5 \cdot 10^2$
vehicle	0.190	=	0.225	$5.4 \cdot 10^7$	>	$2.5 \cdot 10^6$	$5.2 \cdot 10^3$	>	$4.8 \cdot 10^2$
pima-indians	0.228	=	0.234	$3.3 \cdot 10^7$	>	$1.6 \cdot 10^5$	$7.0 \cdot 10^3$	>	$1.3 \cdot 10^2$
sonar	0.144	<	0.288	$1.6 \cdot 10^8$	>	$9.1 \cdot 10^5$	$5.0 \cdot 10^3$	>	$5.0 \cdot 10^1$

Table 72: The performance of classification via regression using MDLCIPER as compared to using linear regression (**LR**) and model trees (**MT**).

dataset	MDLCIPER	t-test	**LR**	t-test	**MT**
mushroom	0.000	$<$	0.042	$=$	0.000
vote	0.060	$=$	0.043	$=$	0.039
glass	0.322	$=$	0.425	$=$	0.285
segment	0.080	$<$	0.167	$>$	0.030
anneal	0.124	$>$	0.035	$>$	0.016
sick	0.065	$=$	0.062	$=$	0.062
zoo	0.238	$<$	0.693	$<$	0.574
ionosphere	0.077	$<$	0.151	$=$	0.123
cleveland-14	0.179	$=$	0.159	$=$	0.162
german credit	0.276	$=$	0.235	$=$	0.251
lymphography	0.405	$>$	0.142	$>$	0.176
hepatitis	0.229	$=$	0.212	$=$	0.212
autos	0.430	$=$	0.383	$=$	0.228
hungarian-14	0.241	$=$	0.176	$=$	0.192
vowel	0.397	$<$	0.568	$>$	0.185
heart-statlog	0.185	$=$	0.167	$=$	0.207
balance-scale	0.090	$=$	0.136	$=$	0.122
vehicle	0.225	$=$	0.255	$=$	0.188
pima-indians	0.234	$=$	0.228	$=$	0.227
sonar	0.288	$=$	0.260	$=$	0.226

Table 73: The performance of classification via regression using CVCIPER as compared to using linear regression (**LR**) and model trees (**MT**).

dataset	CVCIPER	t-test	**LR**	t-test	**MT**
mushroom	0.000	$<$	0.042	$=$	0.000
vote	0.050	$=$	0.043	$=$	0.039
glass	0.327	$=$	0.425	$=$	0.285
segment	0.060	$<$	0.167	$=$	0.030
anneal	0.018	$=$	0.035	$=$	0.016
sick	0.063	$=$	0.062	$=$	0.062
zoo	0.059	$<$	0.693	$<$	0.574
ionosphere	0.060	$<$	0.151	$=$	0.123
cleveland-14	0.172	$=$	0.159	$=$	0.162
german credit	0.244	$=$	0.235	$=$	0.251
lymphography	0.209	$=$	0.142	$=$	0.176
hepatitis	0.186	$=$	0.212	$=$	0.212
autos	0.212	$<$	0.383	$=$	0.228
hungarian-14	0.215	$=$	0.176	$=$	0.192
vowel	0.298	$<$	0.568	$>$	0.185
heart-statlog	0.193	$=$	0.167	$=$	0.207
balance-scale	0.085	$=$	0.136	$=$	0.122
vehicle	0.190	$<$	0.255	$=$	0.188
pima-indians	0.228	$=$	0.228	$=$	0.227
sonar	0.144	$=$	0.260	$=$	0.226

Table 74: The performance of classification via regression using MDLCIPER as comapred to the performance of decision trees (**J48**); support vector machines: **SMO-E1**, **SMO-E2**, **SMO-KR**; and Naive Bayes (**NB**).

dataset	MDLCIPER	t-test	**J48**	t-test	**SMO-E1**	t-test	**SMO-E2**	t-test	**SMO-KR**	t-test	**NB**
mushroom	0.000	=	0.000	=	0.000	=	0.000	=	0.001	<	0.042
vote	0.060	=	0.050	=	0.039	=	0.071	=	0.060	=	0.100
glass	0.322	=	0.322	=	0.425	=	0.397	<	0.645	<	0.523
segment	0.080	>	0.037	=	0.072	>	0.053	<	0.182	<	0.197
anneal	0.124	>	0.013	>	0.041	>	0.018	=	0.090	=	0.140
sick	0.065	=	0.062	=	0.062	=	0.064	=	0.062	=	0.076
zoo	0.238	=	0.079	=	0.040	=	0.030	=	0.277	=	0.050
ionosphere	0.077	=	0.105	=	0.125	=	0.091	<	0.242	<	0.174
cleveland-14	0.179	=	0.240	=	0.169	=	0.209	=	0.169	=	0.172
german credit	0.276	=	0.291	=	0.247	=	0.308	=	0.300	=	0.247
lymphography	0.405	>	0.216	>	0.128	>	0.162	>	0.196	>	0.176
hepatitis	0.229	=	0.229	=	0.195	=	0.220	=	0.178	=	0.169
autos	0.430	>	0.238	=	0.306	>	0.187	=	0.528	=	0.466
hungarian-14	0.241	=	0.245	=	0.203	=	0.211	=	0.188	=	0.176
vowel	0.397	>	0.186	>	0.295	>	0.025	<	0.697	=	0.372
heart-statlog	0.185	=	0.215	=	0.167	=	0.196	=	0.174	=	0.167
balance-scale	0.090	<	0.232	=	0.128	=	0.083	=	0.115	<	0.554
vehicle	0.225	<	0.297	=	0.254	=	0.194	<	0.593	=	0.241
pima-indians	0.234	=	0.257	=	0.232	=	0.224	<	0.349	=	0.317
sonar	0.288	=	0.322	=	0.216	>	0.168	=	0.313	<	0.042

Table 75: The performance of classification via regression using CVCIPER as comapred to the performance of decision trees (**J48**); support vector machines: **SMO-E1**, **SMO-E2**, **SMO-KR**; and Naive Bayes (**NB**).

dataset	CVCIPER	t-test	**J48**	t-test	**SMO-E1**	t-test	**SMO-E2**	t-test	**SMO-KR**	t-test	**NB**
mushroom	0.000	=	0.000	=	0.000	=	0.000	=	0.001	<	0.042
vote	0.050	=	0.050	=	0.039	=	0.071	=	0.060	<	0.100
glass	0.327	=	0.322	=	0.425	=	0.397	<	0.645	<	0.523
segment	0.060	=	0.037	=	0.072	=	0.053	<	0.182	<	0.197
anneal	0.018	=	0.013	=	0.041	=	0.018	<	0.090	<	0.140
sick	0.063	=	0.062	=	0.062	=	0.064	=	0.062	<	0.076
zoo	0.059	=	0.079	=	0.040	=	0.030	<	0.277	=	0.050
ionosphere	0.060	=	0.105	=	0.125	=	0.091	<	0.242	<	0.174
cleveland-14	0.172	=	0.240	=	0.169	=	0.209	=	0.169	=	0.172
german credit	0.244	<	0.291	=	0.247	<	0.308	<	0.300	=	0.247
lymphography	0.209	=	0.216	=	0.128	=	0.162	=	0.196	=	0.176
hepatitis	0.186	=	0.229	=	0.195	=	0.220	=	0.178	=	0.169
autos	0.212	=	0.238	=	0.306	=	0.187	<	0.528	<	0.466
hungarian-14	0.215	=	0.245	=	0.203	=	0.211	=	0.188	=	0.176
vowel	0.298	>	0.186	=	0.295	>	0.025	<	0.697	<	0.372
heart-statlog	0.193	=	0.215	=	0.167	=	0.196	=	0.174	=	0.167
balance-scale	0.085	<	0.232	=	0.128	=	0.083	=	0.115	<	0.554
vehicle	0.190	<	0.297	<	0.254	=	0.194	<	0.593	=	0.241
pima-indians	0.228	=	0.257	=	0.232	=	0.224	<	0.349	<	0.317
sonar	0.144	<	0.322	=	0.216	=	0.168	<	0.313	<	0.042

Table 76: The performance of classification via regression using CVCIPERX (MDL) and MDLCIPERX (CV).

		ce			*ssc*			*mcl*	
dataset	MDL	t-test	CV	MDL	t-test	CV	MDL	t-test	CV
mushroom	0.000	=	0.000	$1.4 \cdot 10^{7}$	>	$1.7 \cdot 10^{6}$	$7.4 \cdot 10^{2}$	>	$1.1 \cdot 10^{2}$
vote	0.050	=	0.060	$1.5 \cdot 10^{7}$	>	$3.1 \cdot 10^{5}$	$3.2 \cdot 10^{3}$	>	$2.1 \cdot 10^{2}$
glass	0.304	=	0.355	$1.9 \cdot 10^{8}$	>	$5.3 \cdot 10^{6}$	$1.6 \cdot 10^{4}$	>	$9.9 \cdot 10^{2}$
segment	0.071	<	0.093	$5.8 \cdot 10^{7}$	>	$3.6 \cdot 10^{6}$	$2.5 \cdot 10^{3}$	>	$3.0 \cdot 10^{2}$
anneal	0.019	<	0.136	$5.9 \cdot 10^{7}$	>	$9.7 \cdot 10^{5}$	$2.3 \cdot 10^{3}$	>	$7.1 \cdot 10^{1}$
sick	0.063	=	0.065	$5.7 \cdot 10^{7}$	>	$4.7 \cdot 10^{5}$	$5.1 \cdot 10^{3}$	>	$1.6 \cdot 10^{2}$
zoo	0.059	=	0.238	$2.3 \cdot 10^{7}$	>	$1.1 \cdot 10^{6}$	$1.2 \cdot 10^{3}$	>	$8.0 \cdot 10^{1}$
ionosphere	0.085	=	0.077	$1.2 \cdot 10^{8}$	>	$4.1 \cdot 10^{6}$	$3.6 \cdot 10^{3}$	>	$1.8 \cdot 10^{2}$
cleveland-14	0.169	=	0.169	$9.0 \cdot 10^{7}$	>	$6.2 \cdot 10^{5}$	$6.3 \cdot 10^{3}$	>	$1.1 \cdot 10^{2}$
german credit	0.246	<	0.282	$7.6 \cdot 10^{7}$	>	$1.4 \cdot 10^{6}$	$2.6 \cdot 10^{3}$	>	$9.8 \cdot 10^{1}$
lymphography	0.216	=	0.419	$2.1 \cdot 10^{8}$	>	$4.0 \cdot 10^{5}$	$6.5 \cdot 10^{3}$	>	$4.7 \cdot 10^{1}$
hepatitis	0.186	=	0.229	$8.8 \cdot 10^{7}$	>	$3.6 \cdot 10^{3}$	$8.3 \cdot 10^{3}$	>	$1.5 \cdot 10^{1}$
autos	0.254	<	0.446	$1.2 \cdot 10^{8}$	>	$1.8 \cdot 10^{6}$	$4.1 \cdot 10^{3}$	>	$1.4 \cdot 10^{2}$
hungarian-14	0.207	=	0.234	$5.4 \cdot 10^{7}$	>	$2.7 \cdot 10^{5}$	$6.0 \cdot 10^{3}$	>	$9.3 \cdot 10^{1}$
vowel	0.298	<	0.397	$3.5 \cdot 10^{7}$	>	$1.8 \cdot 10^{6}$	$2.2 \cdot 10^{3}$	>	$3.8 \cdot 10^{2}$
heart-statlog	0.159	=	0.196	$8.1 \cdot 10^{7}$	>	$3.3 \cdot 10^{5}$	$5.9 \cdot 10^{3}$	>	$8.2 \cdot 10^{1}$
balance-scale	0.094	=	0.085	$4.7 \cdot 10^{7}$	>	$1.2 \cdot 10^{6}$	$7.1 \cdot 10^{3}$	>	$4.9 \cdot 10^{2}$
vehicle	0.193	<	0.235	$5.2 \cdot 10^{7}$	>	$3.0 \cdot 10^{6}$	$4.7 \cdot 10^{3}$	>	$4.6 \cdot 10^{2}$
pima-indians	0.234	=	0.233	$3.3 \cdot 10^{7}$	>	$1.6 \cdot 10^{5}$	$6.8 \cdot 10^{3}$	>	$1.2 \cdot 10^{2}$
sonar	0.144	<	0.284	$1.7 \cdot 10^{8}$	>	$9.9 \cdot 10^{5}$	$4.8 \cdot 10^{3}$	>	$4.9 \cdot 10^{1}$

Table 77: The performance (in terms of *ce*) of classification via regression using MDLCIPERX, linear regression (**LR**) and model trees (**MT**).

dataset	MDLCIPERX	t-test	**LR**	t-test	**MT**
mushroom	0.000	<	0.042	=	0.000
vote	0.060	=	0.043	=	0.039
glass	0.355	=	0.425	=	0.285
segment	0.093	<	0.167	>	0.030
anneal	0.136	>	0.035	>	0.016
sick	0.065	=	0.062	=	0.062
zoo	0.238	<	0.693	<	0.574
ionosphere	0.077	<	0.151	=	0.123
cleveland-14	0.169	=	0.159	=	0.162
german credit	0.282	>	0.235	=	0.251
lymphography	0.419	>	0.142	>	0.176
hepatitis	0.229	=	0.212	=	0.212
autos	0.446	=	0.383	=	0.228
hungarian-14	0.234	=	0.176	=	0.192
vowel	0.397	<	0.568	>	0.185
heart-statlog	0.196	=	0.167	=	0.207
balance-scale	0.085	=	0.136	=	0.122
vehicle	0.235	=	0.255	=	0.188
pima-indians	0.233	=	0.228	=	0.227
sonar	0.284	=	0.260	=	0.226

Table 78: The performance (in terms of *ce*) of classification via regression using CVCIPERX, linear regression (**LR**) and model trees (**MT**).

dataset	CVCIPERX	t-test	**LR**	t-test	**MT**
mushroom	0.000	<	0.042	=	0.000
vote	0.050	=	0.043	=	0.039
glass	0.304	=	0.425	=	0.285
segment	0.071	<	0.167	>	0.030
anneal	0.019	=	0.035	=	0.016
sick	0.063	=	0.062	=	0.062
zoo	0.059	<	0.693	<	0.574
ionosphere	0.085	<	0.151	=	0.123
cleveland-14	0.169	=	0.159	=	0.162
german credit	0.246	=	0.235	=	0.251
lymphography	0.216	=	0.142	=	0.176
hepatitis	0.186	=	0.212	=	0.212
autos	0.254	=	0.383	=	0.228
hungarian-14	0.207	=	0.176	=	0.192
vowel	0.298	<	0.568	>	0.185
heart-statlog	0.159	=	0.167	=	0.207
balance-scale	0.094	=	0.136	=	0.122
vehicle	0.193	<	0.255	=	0.188
pima-indians	0.234	=	0.228	=	0.227
sonar	0.144	=	0.260	=	0.226

Table 79: The performance (in terms of *ce*) of classification via regression using MDLCIPERX as compared to the performance of (**J48**); support vector machines: **SMO-E1**, **SMO-E2**, **SMO-KR**; and Naive Bayes (**NB**).

dataset	MDLCIPERX	t-test	**J48**	t-test	**SMO-E1**	t-test	**SMO-E2**	t-test	**SMO-KR**	t-test	**NB**
mushroom	0.000	=	0.000	=	0.000	=	0.000 =	0.001	<	0.042	
vote	0.060	=	0.050	=	0.039	=	0.071 =	0.060	=	0.100	
glass	0.355	=	0.322	=	0.425	=	0.397 <	0.645	<	0.523	
segment	0.093	>	0.037	=	0.072	>	0.053 <	0.182	<	0.197	
anneal	0.136	>	0.013	>	0.041	>	0.018 =	0.090	=	0.140	
sick	0.065	=	0.062	=	0.062	=	0.064 =	0.062	=	0.076	
zoo	0.238	=	0.079	=	0.040	=	0.030 =	0.277	=	0.050	
ionosphere	0.077	=	0.105	=	0.125	=	0.091 <	0.242	<	0.174	
cleveland-14	0.169	=	0.240	=	0.169	=	0.209 =	0.169	=	0.172	
german credit	0.282	=	0.291	=	0.247	=	0.308 =	0.300	=	0.247	
lymphography	0.419	=	0.216	>	0.128	>	0.162 >	0.196	>	0.176	
hepatitis	0.229	=	0.229	=	0.195	=	0.220 =	0.178	=	0.169	
autos	0.446	>	0.238	=	0.306	>	0.187 =	0.528	=	0.466	
hungarian-14	0.234	=	0.245	=	0.203	=	0.211 =	0.188	=	0.176	
vowel	0.397	>	0.186	>	0.295	>	0.025 <	0.697	=	0.372	
heart-statlog	0.196	=	0.215	=	0.167	=	0.196 =	0.174	=	0.167	
balance-scale	0.085	<	0.232	=	0.128	=	0.083 =	0.115	<	0.554	
vehicle	0.235	<	0.297	=	0.254	=	0.194 <	0.593	=	0.241	
pima-indians	0.233	=	0.257	=	0.232	=	0.224 <	0.349	=	0.317	
sonar	0.284	=	0.322	=	0.216	>	0.168 =	0.313	<	0.042	

Table 80: The performance (in terms of *ce*) of classification via regression using CVCIPERX as compared to the performance of (**J48**); support vector machines: **SMO-E1**, **SMO-E2**, **SMO-KR**; and Naive Bayes (**NB**)

dataset	CVCIPERX	t-test	**J48**	t-test	**SMO-E1**	t-test	**SMO-E2**	t-test	**SMO-KR**	t-test	**NB**
mushroom	0.000	=	0.000	=	0.000	=	0.000	=	0.001	<	0.042
vote	0.050	=	0.050	=	0.039	=	0.071	=	0.060	<	0.100
glass	0.304	=	0.322	=	0.425	=	0.397	<	0.645	<	0.523
segment	0.071	>	0.037	=	0.072	=	0.053	<	0.182	<	0.197
anneal	0.019	=	0.013	=	0.041	=	0.018	<	0.090	<	0.140
sick	0.063	=	0.062	=	0.062	=	0.064	=	0.062	<	0.076
zoo	0.059	=	0.079	=	0.040	=	0.030	<	0.277	=	0.050
ionosphere	0.085	=	0.105	=	0.125	=	0.091	<	0.242	<	0.174
cleveland-14	0.169	=	0.240	=	0.169	=	0.209	=	0.169	=	0.172
german credit	0.246	<	0.291	=	0.247	<	0.308	<	0.300	=	0.247
lymphography	0.216	=	0.216	=	0.128	=	0.162	=	0.196	=	0.176
hepatitis	0.186	=	0.229	=	0.195	=	0.220	=	0.178	=	0.169
autos	0.254	=	0.238	=	0.306	=	0.187	<	0.528	=	0.466
hungarian-14	0.207	=	0.245	=	0.203	=	0.211	=	0.188	=	0.176
vowel	0.298	>	0.186	=	0.295	>	0.025	<	0.697	<	0.372
heart-statlog	0.159	=	0.215	=	0.167	=	0.196	=	0.174	=	0.167
balance-scale	0.094	<	0.232	=	0.128	=	0.083	=	0.115	<	0.554
vehicle	0.193	<	0.297	<	0.254	=	0.194	<	0.593	=	0.241
pima-indians	0.234	=	0.257	=	0.232	=	0.224	<	0.349	<	0.317
sonar	0.144	<	0.322	=	0.216	=	0.168	<	0.313	<	0.042

C List of Publications

List of publications related to this dissertation.

Original scientific articles

- Pečkov, A.; Džeroski, S.; Todorovski L. A minimal description length scheme for polynomial regression. *Lecture Notes in Computer Science* **5012**, 284–295 (2008).
- Laganis, J.; Pečkov, A.; Debeljak, M. Modeling radial growth increment of black alder (Alnus glutionsa (L.) Gaertn.) tree. *Ecological Modelling* **215**, 180–189 (2008).

Published scientific conference and workshop papers

- Pečkov, A.; Džeroski, S.; Todorovski, L.; Ljubič, P. Improving the heuristic solver for polynomial equations - CIPER. In: *Proc. Second Balkan Conference on Informatics.* 397–404 (Institute of Informatics, Faculty of Natural Sciences and Mathematics, Skopje, Macedonia, 2005).
- Pečkov, A.; Todorovski, L.; Džeroski, S. Proper versus ad-hoc MDL principle for polynomial regression. In: *Proc. Ninth International Multi-Conference Information Society.* 263–266 (Jožef Stefan Institute, Ljubljana, Slovenia, 2006).
- Pečkov, A.; Todorovski, L.; Džeroski, S. Multitarget polynomial regression with constraints. In: *Proc. International Workshop on Constraint-Based Mining and Learning (held at ECML/PKDD 2007).* 61–72 (Warsaw University of Technology, Warsaw, Poland, 2007).
- Ristov, S.; Pečkov, A. Machine Learning Approach for Early Detection of Cardiovascular Deceases (CVD). In: *Web Proc. 2nd ICT Innovations Conference.* 41–50 (ICT ACT, Skopje, Macedonia, 2010).

D Biography

Aleksandar Pečkov was born in Skopje, Macedonia, on February 27, 1980. He completed the Bachelor of Science degree in theoretical mathematics at the Faculty of Natural Sciences and Mathematics, Ss. Kiril and Methodius University, Skopje, Macedonia in 2004. Afterwards, in 2005 he enrolled at the PhD program New Media and E-science at the Jožef Stefan International Postgraduate School. During his high-school and student days he successfully attended many mathematical and programing contests. During his PhD studies he collaborated with many colleagues on different research projects.