DISCOVERY OF NUMERICAL DEPENDENCIES IN FORM OF RATIONAL EXPRESSIONS.

Mikhail V.Kiselev, Sergei B.Arseniev
Computer Patient Monitoring Laboratory at National Research Center of Surgery, per.Abrikosovski, 2, Moscow 119874, RUSSIA
e-mail: megaputer@glas.apc.org phone, fax 7(095)485 9354

A B S T R A C T

Discovery of numerical dependencies formalized as algebraic equations or other symbolic laws is an important class of KDD problems. The major part of problems solved by the machine discovery system PolyAnalyst belongs to this class [1, 2] . Searching for structure hidden in data PolyAnalyst builds and tests hypotheses about interdependencies or other regularities in data in a form of functional programs constructed recursively from simpler programs using 4 production methods. Two of them, 'rational expression' and 'functional composition', are most important for discovery of numerical dependencies. An efficient search strategy based mainly on the rational expression productions is discussed in this paper. Procedures built by PolyAnalyst are treated as regression models which are nonlinear in general case. Specific methods of solution of associated nonlinear regression problems are described.

KEYWORDS: knowledge discovery, nonlinear regression, rational expressions

1. INTRODUCTION

Problems related to discovery of numerical dependencies between real valued variables are at least as common for science and industry as classification problems. Until recent times the prevailing number of works in this field was devoted to either sub-symbolic approaches like neural nets or methods based on regression models given a priori or selected from some narrow class. Only in last decade several systems were developed which formalize discovered dependencies as mathematical equations of a very general form. The well known systems of this class are BACON [3],

ABACUS [4], FAHRENHEIT [5], E*[6], KEPLER [7], 49er [8]. The key factor determining performance of these systems is a compromise between computational complexity and variety of forms of equations which can be discovered. This compromise is achieved due to choice of more or less narrow set of formulae in which search is concentrated. For example, the Equation Finder, function-finding module of 49er system [9] searches for the best formula expressing dependence of variable y on variable x among equations of the form $\sum a_i f_i(x,y) = F(x,y)$.

In the KDD system PolyAnalyst [10, 11] designed by us rational expressions (polynoms divided by polynoms) play such a special role. In the present paper we consider search strategy based on this choice, describe methods of solution of nonlinear regression problems specific for this approach, and discuss its pluses and minuses. To make the consideration clearer we begin with a very brief outline of PolyAnalyst architecture.

2. POLYANALYST: SEARCH IN THE SPACE OF FUNCTIONAL PROGRAMS.

PolyAnalyst analyzes data represented as a set of records consisting of one or several fields so that all records have the same format. It can discover dependence of some field on other fields and express it (if it exists) in explicit symbolic form. Beside discovery of numerical dependencies this class of problems includes also classification and problems close to grammar inference.

The kernel of the data mining system PolyAnalyst is a mechanism which builds new functional programs from existing functional programs. Not considering semantics of these programs they can be understood as abstract objects with some number of inputs (or without inputs) and 1 output. Inputs (also called arguments) and outputs are marked by their **type** and some other attributes. The simplest atomic functional procedures are called primitives. The set of primitives is determined by structure and properties of data to be analyzed. The set of primitives includes standard and user-defined primitives. Information in databases is accessed via special data access primitives which also can be standard and user-defined. Such an architecture gives PolyAnalyst flexibility because the user-defined data access primitives can issue SQL requests, read data files or support more sophisticated data access not bound to the relational database ideology.

For building new functional programs from existing ones PolyAnalyst uses four production schemes. The simplest scheme is **functional composition**. PolyAnalyst takes one functional program (it is called producing function) and connects some its inputs with outputs of some other existing programs. This process is controlled by the **compatibility rules** which do not permit certain combinations of attributes of connected inputs and outputs. The second scheme serves to realize such often used

concepts as ∃ and ∀ quantors, calculation of average, integration, etc. To form a new functional program using this scheme PolyAnalyst selects some existing program returning numerical or boolean value. The new program returns sum (or result of boolean OR operation) of values returned by the old procedure which correspond to all possible combinations of values of certain old procedure's arguments. The third scheme realizes an analogue of **while** construction of the conventional programming languages like C or PASCAL. At last the scheme most important for discovery of the numerical laws produces functional programs realizing rational expressions (a polynom divided by a polynom) formed from numerical constants and some existing programs (naturally, they should return numerical values). This scheme will be considered in details in the next section.

In fact, this mechanism realizes a simple universal programming language suitable for formalization of wide range of laws and rules which can be discovered in data. For example, **if** construction can be expressed using functional composition with the special primitive called **TF-commutator** taken as a producing function. First argument of the TF-commutator is boolean. If it is **true** then result of the TF-commutator is equal to the value of it's second argument, else it is equal to the value of the third argument. PolyAnalyst has the explanation generator module which translates rules expressed in form of functional programs into a clear verbal form including standard mathematical notation.

The generator of functional programs is controlled by the search strategy module. The search direction is determined in accordance with evaluation of each individual functional program carried out by the search strategy module. In case of discovery of the numerical laws the system uses two evaluation criteria. These are the standard error of the respective regression model and the entropy of values returned by the functional program on the set of training examples. The search process is a combination of full search (low priority component) and so-called 'generalizing transformations' - GT (high priority component). The GT process takes one of the best found programs (called root program) and uses it for creation of new programs with help of one of the above mentioned production schemes in all possible ways satisfying the following condition. Every derivative program should have some set of arguments such that when these arguments have certain constant values the derivative program becomes identical to the root program. This condition guarantees that the derivative programs fit the data not worse (in terms of the standard error) than the root program. In normal situations the GT process takes the most part of PolyAnalyst computation time. Realization of full and GT search in the space of rational expressions is considered in the next section.

In conclusion of this section we say some words about computational complexity of our approach. In our case exact evaluation of the computational complexity is a complex problem because of its very strong dependence on quality of explored data. When our algorithm is tested on random data without any interdependencies between

attributes its computation time depends on complexity of built procedures exponentially. If there exists strong dependence between attributes the system detects its presence since some of built procedures have high evaluation in terms of the above mentioned criteria. In this case PolyAnalyst starts multi-level GT search and dependence of computation time on procedure complexity becomes close to linear. More detailed consideration of PolyAnalyst's computation time for data with various statistical characteristics would require too much place so that we here limit ourselves by this short remark. Reasonable solution time of various real-world data analysis problems also can be considered as an evidence of acceptable computational complexity of PolyAnalyst's algorithms.

3. SEARCH IN THE SPACE OF RATIONAL EXPRESSIONS.

Fisrt of all we give some reasons why rational expressions represent the special construction of PolyAnalyst's internal language.

1. Rational expressions form a very large set of expressions with easily understandable properties, convenient for automated and manual transformations. Identity of two rational expressions can be easily detected automatically.

2. They can be evaluated quickly and exactly in comparison with, say, logarithms or trigonometric functions.

3. Our experience shows that rational expressions describe various data very well except cases of logarithmic and exponential dependencies in their asymptotic region or explicit periodical dependencies. Variations of denominator near zero allows to model fast changes of dependent variable, peaks, discontinuities and other strong nonlinear effects.

4. Coefficients entering rational expression considered as a regression model can be calculated much more easily and exactly than for a general nonlinear regression model.

At last it should be noted that the choice of rational expressions as a special production method does not forbid PolyAnalyst building functions of a more general form. The set of functions which can be built depends also on the set of functional primitives. For example, if we have the binary primitive $pow(x,y) = x^y$ we can build the expressions Ax^yz or $AB^{1/(x^2+C)}$.

PolyAnalyst produces rational expressions which can include all existing functional programs returning numerical values except those that are rational expressions themselves. The order in which rational expressions are built is determined by their complexity. The complexity of rational expression is a sum of complexities of functional programs entering it and its own complexity which is proportional to the number of algebraic operators in it. The full search is organized

as a breadth-first search process which runs consecutively through the subsets of rational expressions with increasing complexity.

The main problem which has been solved on the stage of implementation of rational expression production mechanism in PolyAnalyst was to design it in a way that would give the minimum number of equivalent rational expressions. The definitions of normalized form of rational expression and ordering relation on the set of the normalized rational expressions help to achieve this goal. All normalized rational expressions are defined by

a. the number of functional programs entering them (N_e);

b. the number of terms in the numerator (N_n);

c. the number of terms in the denominator minus 1 (N_d) (polynoms are considered as rational expressions with N_d = 0; their denominator consists of one term equal to 1);

d. the polynom representing the numerator (**N**);

e. the first term of the denominator ($\mathbf{d_0}$);

f. the polynom representing the rest terms of the denominator (**D**).

The first term $\mathbf{d_0}$ of the denominator is considered as having no multiplicative coefficient. By that the scale of all the multiplicative coefficients in rational expression is fixed. It is why $\mathbf{d_0}$ term is separated from other denominator's terms. The components *a*.-*f*. defining rational expression are enumerated in order of descending significance relatively to the above mentioned order on the set of normalized rational expressions. The lexicographic order is defined on the set of polynoms. It is determined by their terms so that the first term is considered as most significant. Terms entering the polynoms **N** and **D** are ordered so that the first term has the highest order. The first term of the polynom **D** should not have the order greater than the order of $\mathbf{d_0}$. The set of terms entering the rational expression also has the lexicographic order defined on it. The order is determined by position of the factors (functional programs) constituting the term in the list of all functional programs entering the rational expression. The factors constituting the terms are ordered so that the first factor has the highest order. Although search in the set of normalized forms of rational expressions does not avoid all equivalencies many equivalencies such as presence of common factor in numerator and denominator can be easily detected by the additional analysis of the structure representing rational expression constructed.

The large class of the generalizing transformations of rational expressions can be defined due to the following obvious fact. If **A**, **B**, $\mathbf{P_1}$, $\mathbf{P_2}$, $\mathbf{P_3}$ are polynoms and the rational expression **R** = **A**/ **B** is taken as a root program then all rational expressions $\mathbf{R'} = \dfrac{\mathbf{A}*\mathbf{P_1} + \mathbf{P_2}}{\mathbf{A}*\mathbf{P_1} + \mathbf{P_3}}$ are its GT-derivatives. Indeed, when all coefficients of polynoms $\mathbf{P_2}$ and $\mathbf{P_3}$ are equal to zero $\mathbf{R'}$ becomes identically equal to **R**.

The GT search in the space of rational functions is organized in the following way. The GT search is realized by several concurrent processes p_i. Root of p_0 process

is the best procedure found by the full search process. Root of p_i process ($i \geq 1$) is the best procedure found by the process p_{i-1}. Every p_i process performs the breadth-first search building consecutively the GT-derivatives of its root procedure in order of increasing complexity. If the process p_i finds a new best procedure it destroys all processes p_j for $j>i$ and creates new process p_{i+1} with this procedure taken as a root procedure. The last p_i process in this chain creates its child process p_{i+1} when it finds a GT-derivative of its root procedure whose coefficients all have the T ratio greater than 1. The T ratio of coefficient of regression model is the ratio of its computed value to its standard deviation.

An example of GT search combined with full search is given in section 5. This mechanism is illustrated there by shortened PolyAnalyst search protocol for an empirical law discovery problem from geophysics.

As it was said every built rational expression is considered as regression model. The next section is devoted to the methods of solution of associated regression problems.

4. SOLUTION OF REGRESSION PROBLEMS ASSOCIATED WITH CONSTRUCTED PROCEDURES

After PolyAnalyst has built a new procedure it tries to use it to express dependence of selected variable on other variables with minimum standard error varying values of constants entering the procedure. Since PolyAnalyst searches for the best fit function in a very wide space of procedures it must be able to solve a great variety of nonlinear regression problems. The main difficulty of these regression problems is caused by the fact that in general case there is no a priori information about local and global behavior of the regression function. Implementing PolyAnalyst's regression algorithm we had to solve the following set of problems.

1. Not having any assumptions about global behavior of regression function we risk to find not global minimum of standard error but some local minimum. Figurally speaking we may slide down to some little "pit" thus missing really lowest point on the graph of dependence of standard error on values of regression coefficients.

2. Efficient minimization algorithm cannot be realized without information about characteristic scales of regression coefficients. For example, if some constant entering regression function is a power degree its variation by one tenth may change the standard error as strongly as variation of value of other constant by several thousands. In fact, realizing our algorithm we have made an assumtion about characteristic scale of regression coefficients. Namely, we assume that they lay in the interval 10^{-5} - 10^{5}. Since PolyAnalyst normalizes explored data trying to make all of them of order 10^{0} - 10^{1} this assumtion seems to be reasonable in most cases.

3. Local behavior of regression function may be quite unusual. For example, assume that explored data has a form of time sequences of values a_i and regression function is *f = A * (the number of measurements i for which* a_i *> B) + C*. It is clear that the characteristic surface of that function consists of horizontal "steps": $\frac{\partial std.err}{\partial B}$ either equals to zero or does not exist.

4. Practical realization of algorithm solving regression problems for so wide variety of nonlinear models on the real hardware (Intel 87 family math co-processor in our case) faces numerous problems caused by over- and underflows, exceptions, domain errors, roundoffs, and many other situations.

Obviuosly, the nonlinear regression problems in such a general formulation occur in practical computing rarely. Anyway, all the algorithms known to us use more or less strong assumptions about regression model, require hill climbing starting point and initial step to be given or use other user-supplied information and therefore could hardly be used in PolyAnalyst immediately.

After numerous expriments with test and real data we have selected the following strategy. First we describe it in general and then consider its modifications taking into account the structure of regression function as rational expression. The whole procedure includes the following four steps.

1. Random trials in logarithmic scale. Suppose that the regression model includes *n* paramerters. Then the standard error is a function of *n* variables which can be considered as coordinates in n-dimensional space $\mathbf{R}^n$. The first step is repeated for all combinations of signs of these *n* variables that gives 2^n iterations. For each iteration the respective part of $\mathbf{R}^n$ is parametrized by natural logarithm of the coordinates so that the point with coordinates +1 or -1 is represented as (0,...,0). In this region $m_{log.rand}$ points are randomly selected in accordance with n-dimensional normal distribution with center at (0,...,0) and some dispersion σ (we selected σ = 6 taking into account the data normalization made by PolyAnalyst). For each $m_{log.rand}2^n$ points value of standard error is calculated and the point $\mathbf{p_{logr}}$ which gives minimum standard error is selected.

2. Random trials in linear scale. In this step $\mathbf{R}^n$ space is parametrized linearly by values of the regression coefficients. The point $\mathbf{p_{logr}}$ with coordinates p_{logr_i} determined in the previous step is taken as a center of n-dimensional normal distribution with the dispersion p_{logr_i} corresponding to the *i*th coordinate. Using this distribution $m_{lin.rand}$ points are randomly selected. Again, point $\mathbf{p_{linr}}$ which gives minimum standard error *r* is selected among them. This point serves as a starting point for the subsequent hill climbing procedure.

3. Determination of characteristic scales. In this step the same procedure is repeated for each of *n* variables. Its goal is to find such variation Δ_i of the *i*th variable that would satisfy the following requirements

a. $1 - s_l < \dfrac{std.err(...,p_{linr_i} + \Delta_i,...) - r}{r - std.err(...,p_{linr_i} - \Delta_i,...)} < 1 + s_l$ and

b. $\left|\dfrac{std.err(...,p_{linr_i} + \Delta_i,...) - r}{r}\right| > s_s$ or $\left|\dfrac{std.err(...,p_{linr_i} - \Delta_i,...) - r}{r}\right| > s_s$.

These requirements mean that Δ_i correspond to linear and in the same time significant variations of standard error. Exact understanding of the words "linear" and "significant" is determined by values of s_l and s_s. Usually we set s_l = 0.03, s_s = 0.01. $\lg(\Delta_i)$ are determined by the bisection algorithm starting with the range [-6,5]. If for some variables the values of Δ_i satisfying the requirements above do not exist the values of these variables p_{linr_i} are fixed and not changed while the desired values of the rest variables are determined with help of the gradient hill climbing process (step 4).

4. Gradient hill climbing. Having Δ_i we can compute gradient of standard error and start the gradient hill climbing algorithm for minimization of the standard error. We use the following version of this algorithm. Standard error is minimized on the line going through the starting point in the direction **grad**(std.err) using a fast second order method. In the point of the standard error minimum on this line **grad**(std.err) is again calculated and minimum of standard error on the new search line is sought. This process is terminated when two sequential iterations give the relative decrease of standard error less than 1%.

Tests show that this procedure works stable on various nonlinear regression models and has satisfactory timing characteristics. However if it is known that the regression function is rational then knowledge of its structure as rational expression can make the search many times faster. This knowledge gives the following three benefits:

- Dependence of regression function on coefficients of the polynom in the numerator of rational expression is linear. Therefore, these coefficients can be determined directly for any known values of the rest coefficients by the least squares method. Thus, dimension n of space in which the hill climbing is carried out is decreased by the number of terms in the numerator. Clearly it leads to significant decrease of computation time because computational complexity of step 1 depends on n exponentially, complexity of steps 3 and 4 - linearly.

- During steps 3 and 4 regression function is often calculated for the same values of some coefficients. Knowledge of structure of the rational expression allows to recalculate only those parts which depend on coefficients changed.

- When the GT search is performed in set of rational expressions as it was described in section 3 steps 1 and 2 can be bypassed. In this case the starting point for the hill climbing corresponds to the set of values of the coefficients which make

the GT derivative rational expression identical to its root procedure. It is reasonable because as a rule the corrections made by the generalizing transformations do not change the coordinates of minimum of standard error significantly.

Implementation of the regression algorithm using all the advantages given by representation of regression function as rational expression has made PolyAnalyst several times faster because the part of rational expressions among all procedures built by PolyAnalyst often exceeds 90%.

5. EXAMPLE OF POLYANALYST SEARCH TRACE PROTOCOL.

In order to illustrate operation of PolyAnalyst search strategy module we conclude the present paper by an example related to a real problem solved by PolyAnalyst. The data to be analyzed represent 7215 measurements of logarithm of electron density $log(d_e)$ in ionosphere at various altitudes. Each measurement is accompanied by values of 7 parameters characterizing meteorological and geophysical situation at the moment of measurement. Thus, the problem is to find the empirical law which expresses dependence of $log(d_e)$ on these 8 independent variables (altitude together with 7 other parameters). The independent variables will be denoted as x_0 - x_7.

In process of search PolyAnalyst built many empirical dependencies some of which are shown on Fig.1. Since the present paper is devoted mainly to discovery of empirical laws in a form of rational expressions only the search trace in space of rational expressions is shown (for example, the search path to the **if**-constuction entering the expression D1 is omitted). The leftmost column contains laws found by the full search process. They are shown as a sequence of formulae with increasing complexity linked by vertical arrows. It can be seen that two of them, the first and the last, were taken as starting points for the first level GT search. Three GT-descendants of the procedure A are shown in the second column. The vertical arrows show the order they were constructed by the GT search process. The procedure A3 itself was selected as a root for the second level GT search. The procedure D1 corresonds to the best found empirical law for $log(d_e)$. The laws on the Fig.1 are marked by their standard errors (in percents) and time when they were built. It is seen that thanks to GT search mechanism PolyAnalyst can build and test quite complex laws in reasonable time (the computation was carried out on a Pentium-90 machine).

6. CONCLUSION.

Designing PolyAnayst we aimed to provide it with the following two key features. First, its internal language should have expressive power of universal programming

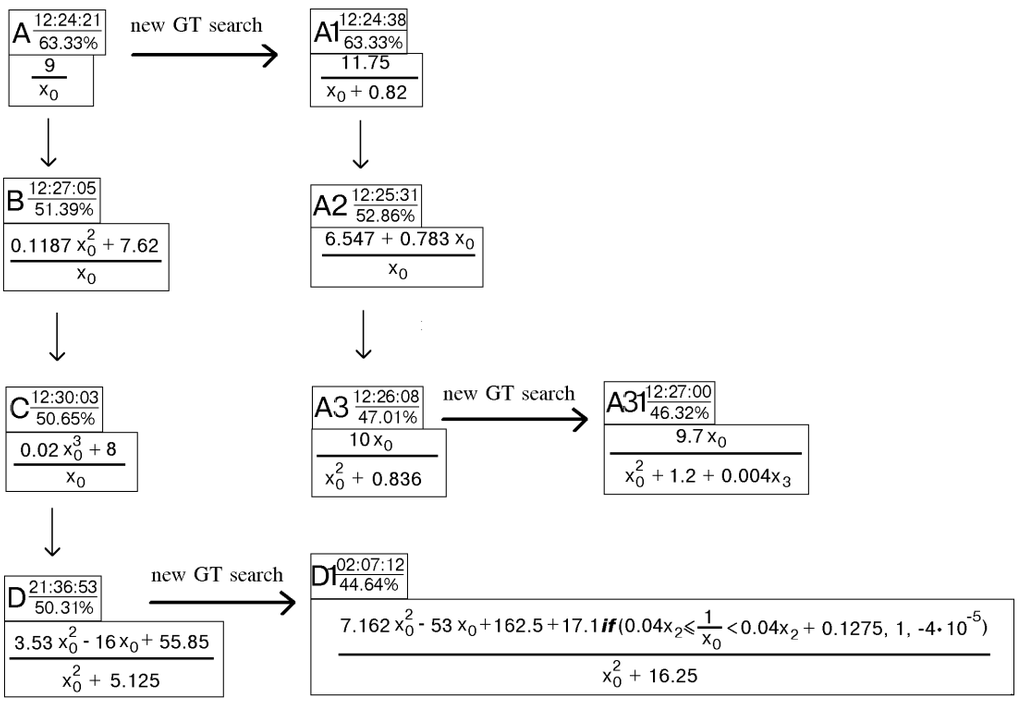

Fig.1 Several empirical laws found by the full and GT search mechasims. Expression **if***(a, b, c)* equals to *b* if condition *a* is true; else it equals to *c*.

language to be able to express all possible forms of discovered dependencies and rules. Second, it should have convenient and flexible mechanisms allowing to concentrate search in some classes of empirical laws which could be selected easily. Applying PolyAnalyst to real problems we found that explored data are often modelled well by rational expressions and therefore in many cases rational expressions can be naturally chosen as a subset including most part of procedures built by PolyAnalyst. Significant decrease of computation time gained from taking into account structure of rational expressions makes us believe that search in the set of rational expressions may give the best ratio *<expressive power>/< computation time >* for many problems.

REFERENCES

[1] Kiselev, M.V., Arseniev, S.B., and Flerov E.V., "PolyAnalyst - a Machine Discovery System for Intelligent Analysis of Clinical Data," ESCTAIC-4 Abstracts (European Society for Computer Technology in Anaesthesiology and Intensive Care), Halkidiki, Greece, p. H6.(1994).

[2] Arseniev, S.B., Kiselev, M.V., and Classen, B., "Patient Ventillation Management Expert Rules derived from Ulm University Clinic Using PolyAnalyst - Knowledge Discovery System," Proceedings of ECML-95 Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases, Heraklion, Greece, pp 199-203. (1995).

[3] Langley, P., Simon, H.A., Bradshaw, G.L., and Zytkow, J.M., "Scientific discovery: Computational explorations of the creative processes." Cambridge, MA: MIT Press., 1987

[4] Falkenhainer, B.C., and Michalski, R.S., "Integrating Quantitative and Qualitative Discovery in the ABACUS System" In: Y.Kodratoff, R.S.Michalski, (Eds.): Machine Learning: An Artificial Intelligence Approach (Volume III). San Mateo, CA: Kaufmann, pp 153-190. (1990).

[5] Zytkow, J.M., and Zhu, J., "Automated Empirical Discovery in a Numerical Space," Proceedings of the Third Chinese Machine Learning Workshop, Harbin, Peoples' Republic of China, pp.1-11. (1991).

[6] Schaffer, C., "A Proven Domain-Independent Scientific Function-Finding Algorithm," Proceedings of AAAI-90, Menlo Park, CA, pp 828-833. (1990).

[7] Wu, Y., and Wang S., "Discovering Knowledge from Observational Data," Proceedings of IJCAI-89 Workshop on Knowledge Discovery in Databases, Detroit, MI, pp.369-377. (1989).

[8] Zytkow, J.M., and Zembowicz R., "Database Exploration in Search of Regularities," Journal of Inteligent Information Systems, Vol.2, pp.39-81. (1993).

[9] Zembowicz, R., and Zytkow J.M., "Discovery of Equations: Experimental Evaluation of Convergence," Proceedings of AAAI-92, Menlo Park, CA, pp 70-75. (1992).

[10] Kiselev, M.V., "PolyAnalyst - a Machine Discovery System Inferring Functional Programs," Proceedings of AAAI Workshop on Knowledge Discovery in Databases'94, Seattle, pp 237-249. (1994).

[11] Kiselev, M.V., "PolyAnalyst 2.0: Combination of Statistical Data Preprocessing and Symbolic KDD Technique," Proceedings of ECML-95 Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases, Heraklion, Greece, pp 187-192. (1995).

[12] Zembowicz, R., and Zytkow J.M., "Testing the Existence of Functional Relationship in Data," Proceedings of AAAI Workshop on Knowledge Discovery in Databases’93, Washington, D.C., pp 102-111. (1993).