Testing the Existence of Functional Relationship in Data

Robert Zembowicz and Jan M. Żytkow
Computer Science Department, Wichita State University, Wichita, KS 67260-0083

Abstract

Discovery of regularities in data involves search in many spaces, for instance in the space of equations, boolean expressions, statistical patterns, and the like. If data do not fit any hypothesis in a particular space, much time could be saved if that space was not searched at all and the search effort was directed to other spaces. A test which determines the non-existence of a solution in a particular space can prevent unneeded search. We concentrate on construction of a test which detects the existence of a functional relationship between two variables in a set of numerical data. The test is passed when the data satisfy the functionality definition. The test is general and computationally simple. It works on data which include error, limited number of outliers, and background noise. Our algorithm detects the functional dependence, but it does not recognize a particular form of that dependence. That task is left to an equation finder, which is called when the data pass the functionality test. We show, how our functionality test works in database exploration within the 49er (read: Forty-Niner) system, as a trigger for the computationally expensive search in the space of equations. Results of tests show the time saved when the test has been applied. Then we discuss how the functionality test, in conjunction with a continuity test can be used to recognize multifunctions, and to split the data so that each part includes a single functional dependency and can be searched for a single equation.

1 Introduction: the role of application conditions

Machine Discovery systems such as BACON (Langley, Simon, Bradshaw & Żytkow 1987), FAHRENHEIT (Żytkow 1987), KDW (Piatetsky-Shapiro & Matheus 1991), EXPLORA (Hoschka & Klösgen 1991; Klösgen 1992), IDS (Nordhausen & Langley 1990), ABACUS (Falkenhainer & Michalski 1986), 49er (Żytkow & Baker 1991; Zembowicz & Żytkow 1993), and many others, explore data in search for hidden knowledge. They conduct search in usually large hypotheses spaces. Although more discoveries can be made when a hypotheses space is larger, search in a large space takes plenty of resources: time and memory. Even if a single search for an equation may take few seconds, when repeated thousands of times it can consume very significant resources. Can unsuccessful search be avoided? What tests can be applied to determine whether there is a chance for a solution in a particular space?

Let us discuss the importance of this problem on the following example, taken from a real-world situation. One of the Navy databases we have analyzed using our database exploration system 49er (Żytkow & Baker 1991; Zembowicz & Żytkow 1992) contains 6032 records, each having values of 35 attributes. 49er, in its first phase of exploration, searches for regularities in two attributes (Żytkow & Zembowicz, 1993). In an exhaustive search and without selecting any subsets in data, 49er would consider $35 \times 34/2 = 595$ pairs of attributes and would try to find a regularity for each pair. If the search for a regularity takes about 1 second per each pair, the system needs about 10 minutes to conduct this search. If the search in *subsets* of data is allowed, the number of hypotheses grows rapidly. In the simplest case, when subsets of data are created by splitting values of a single attribute into two groups (we call it the first level of slicing), 49er would need about 11 hours to try 39270 two-dimensional hypotheses (2×35 subsets $\times\ 34 \times 33/2$ attribute pairs in each subset). On the second level of slicing there are 1256640 *new* hypotheses to try (about 14 days), 25970560 possible regularities on the third level (about 300 days), etc. However, it is quite probable that in most subsets no equations would be found. Even if domain knowledge and search goals reduce the number of hypotheses, the problem remains important, because regularities that have several conditions in their *if* part (each conjunct corresponding to one level of slicing) are typical rather than exceptional in practice.

On a similar task of data analysis, a human would try to use any available knowledge about the attributes and the data to avoid even considering hopeless hypotheses. Therefore our goal is to capture, formalize and implement simple and quick *application tests*, in hope that those tests will prevent unneeded search.

Another reason to avoid the search is when results would not have meaning. Different types of regularities make sense for different types of variables. For instance, even if two variables have numerical values, if the numbers are on the nominal scale rather than on interval or ratio scales, the discovered equations are not meaningful. Considering contingency tables, another class of hypotheses, it does not make sense to aggregate the values of a nominal attribute in two classes to summarize data in a simple contingency table, because in almost all cases a single, a priori conducted grouping has little sense. Simple conditions, which do not even consider the data, but only domain knowledge about the type of variables, can be sufficient. 49er uses each application test before the corresponding space is searched, so that the number of spurious regularities decreases and/or the efficiency of search is improved. Żytkow and Zembowicz (1993) summarize the dependence between variable type and regularity type for the use of 49er.

Now we will concentrate on the functionality test, which is meant to prevent an unsuccessful but costly search in the space of functional expressions. In distinction to tests which consider only the types of variables, functionality test analyzes the actual data.

2 Testing functionality

Empirical equations are efficient summaries of data, if the data distribution is close to a functional relationship. In different machine discovery systems, equations form large, potentially unbound search spaces (Langley *et al.* 1987; Nordhausen & Langley 1990; Falkenhainer & Michalski 1986; Zembowicz & Żytkow 1992, 1993; Moulet 1992). The search in the space of equations is expensive, however, and it may be repeated many times for different combinations of variables and different ranges of data. It would be a good idea to avoid that search on the data which are not conducive to functional description. The problem occurs typically in database discovery, where most of the data follow weak statistical patterns rather than strong functional relationships, but some datasets can be described by equations. Still, when strong functional relationships are possible, it would be a big fault not to notice them when they occur. We need a test which applies to all datasets and is passed only when the data can be approximated by a function. Additionally, the size of databases analyzed by 49er requires a fast test, linear in the size of data: $O(N)$, where N is the number of records.

We will present such a test, which determines a functional dependency between two given variables: independent (called x) and dependent (y). If the dependent variable cannot be identified, the test should be run twice, first for y as dependent and then for x as dependent.

The proposed test *does not* determine the form of functional relation between x and y, but rather tells whether it is impossible to find one. If the test for functionality returns a positive answer, then one could focus on looking for a functional form. There is still no guarantee that a function will be found. The goal of the test is to avoid a search if that search is bound to fail.

2.1 The definition of functionality

We will use the following mathematical definition of functional relationship:

Definition: Given a set DB of pairs (x_i, y_i), $i = 1, \ldots, N$ of two variables x and y, and the range X of x, y is a function of x iff for each x_0 in X, there is exactly one value of y, say y_0, such that (x_0, y_0) is in DB.

If the functions considered in a search space are continuous or almost everywhere continuous, as it is the case in all machine discovery systems, we could add a second requirement: y should be a continuous function of x. However, discrete data make testing continuity difficult and missing data frequently results in artificial discontinuities. After many experiments we decided not to test continuity. The issue of continuity is re-visited in the section on multiple equation finder at the end of this article.

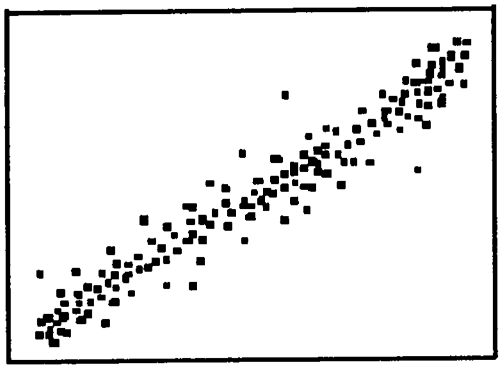
2.2 Problems with functionality in real data

It is easy to check functionality in a given dataset by the strict application of the definition, that is, by checking whether indeed a unique value of y corresponds to one value of x. However, in real world data, because of phenomena collectively called error and noise, strict uniqueness does not occur, so the condition is unrealistic. We want a more relaxed test that will admit data sets which can be reasonably approximated by equations. In particular, we want to admit a deviation in values of y, so rather than require a single value of y, permit a distribution of values in a relatively small interval. This fits a statistical model y $= f(x) + \delta(x)$, where $f(x)$ describes the relationship between x and y, while $\delta(x)$ summarizes the effects of error and noise.

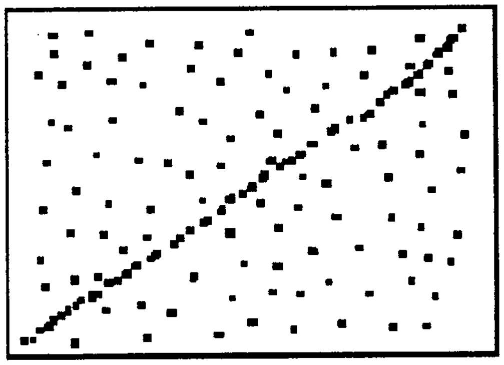
Figure 1a illustrates how the error complicates the problem: for the same value of x there could be many different values of y. A test strictly based on the definition would frequently fail *even* if the data could be described by a function (linear function in the case of data shown in Figure 1a) within a small deviation of y. If the error δ_y of y is known, the comparison of values of y should be related to the error. For example, one can compare $|y_1 - y_2|$ against $\delta_{y_1} + \delta_{y_2}$, where δ_{y_i} is the error of y_i. It means that the definition of the functionality should be modified for our purpose: for each value of x, differences between the corresponding values of y must be in the range $(-\delta_y, \delta_y)$. However, in many cases error is unknown. Typically, error is well estimated only for experimental data.

Figure 1: (a) Effect of error: there is many points that have the same value of x but different y values. These data could be described by a (linear) function plus a small error. (b) Effect of noise: in addition to data that follow a functional dependency, additional distribution of points is not correlated with that dependency, violating the definition of functionality, even if a small error tolerance is permitted. However, a linear dependency in data can be still easily noticed.

(a)



(b)



Another important factor is background noise, which comes from many sources and is common in databases. Even if all combinations of values are present in the data due to the noise, the functional relationship may still stand out by a higher concentration of data. Figure 1b shows background noise added to a functional dependency: in addition to data that can be well described by a linear function, a large number of points are randomly distributed. The functionality test which permits error would fail here because frequently the differences between values of y are large.

Missing data is another factor that adds to the complexity of the problem. For some values of x, data on the functional dependency may not be present, but the background noise may still be there. This is common in databases because in many cases there is no control over data collection process.

Finally, we want to tolerate a small number of outliers, that is data which stand out from noise and lie drastically outside the error distribution. Such data could be entered by error, or they represent a phenomenon which is of potential interest. Still we wish to neglect those data if they are too few to make sense of them.

2.3 Determination of the uniqueness

In databases, typically a small discrete set of values is permitted for each attribute. Given a small set X of x values, and a small set Y of y values, the product $X \times Y$ is computationally manageable, as well as the corresponding frequency table F which is a mapping

$$F : X \times Y \rightarrow N,$$

where N is the set of natural numbers, and $F(x_0, y_0) = n$ when n is the number of datapoints with $x = x_0$ and $y = y_0$.

If the number of distinct values of x and/or y becomes too large to compute the frequency table, the values of x and y can be grouped into bins $b(X)$ and $b(Y)$, respectively. Aggregating the values of x into bins $b(X)$ of equal size Δx means that the point (x_0, y_0) is replaced by a pair of integer numbers (k_x, k_y) such that x_0 is in the range from $x_{min} + k_x \Delta x$ to $x_{min} + (k_x + 1)\Delta x$, and y_0 is in the range from $y_{min} + k_y \Delta y$ to $y_{min} + (k_y + 1)\Delta y$, where x_{min} and y_{min} are the smallest values of x and y, respectively. The frequency table $F(k_x, k_y)$ can be interpreted as a grid $b(X) \times b(Y)$ imposed on the data (x, y) and defined by Δx and Δy.

Binning the original variables x and y into bins $b(X)$ and $b(Y)$ helps to determine functionality in data which include error and/or noise. If the grid size Δy is comparable to the error δ_y, the requirement of maximum difference between y values corresponding to the same value of x can be replaced by: all points in the same x-bin k_x must lie in the adjacent y-bins (for example, in bins $k_y - 1$, k_y, $k_y + 1$). This works only if the bin sizes Δx and Δy are not smaller than corresponding errors; otherwise the functionality test may fail because points in the same x-bin may not lie in *adjacent* y-bins, even if the original data follow a functional dependency plus error. On the other hand, if the sizes Δx and Δy are too large, the test could assign functionality to data that intuitively should not be described by a function. In the extreme case, when Δy is larger than $y_{max} - y_{min}$, all points always lie in the same y-bin, because one bin includes all values of y.

The problem of background noise can be alleviated if one tests the adjacency only for cells that contain an above average number of points. This noise subtraction works effectively only if the background noise is not stronger than the regularity itself. But if the noise is stronger, there is hardly any chance to find a regularity.

If the errors δ_x and δ_y are known, they can be used as the corresponding bin sizes Δx and Δy. But if they are unknown or uncertain, the proper grid sizes must be estimated. Before we present the algorithm which estimates the grid sizes, let us consider an ideal functional dependency, say a straight line, as in Figure 2a. The data are evenly distributed with constant distance Δ in x between points. Let us define the "density" ρ as the average number of points in all cells which contain at least one point. As long as the grid size Δx is smaller than Δ, ρ is equal to one. The density ρ starts to grow when Δx becomes greater than Δ. Note that as opposed to the true density equal to the number of data points divided by the total number of cells, ρ does not depend on the number of x-bins in the following sense. If we extend the range of x by adding more points which are evenly distributed, and keeping Δx constant, then in our ideal example, ρ will have exactly the same values. This is important because it means that the "density" measure ρ does not depend on the range of x or y as long as bin sizes are the same. Therefore ρ can be used to determine Δx and Δy. Figure 2 shows the difference between our parameter ρ and the true density equal to the number of points divided by the total number of cells.

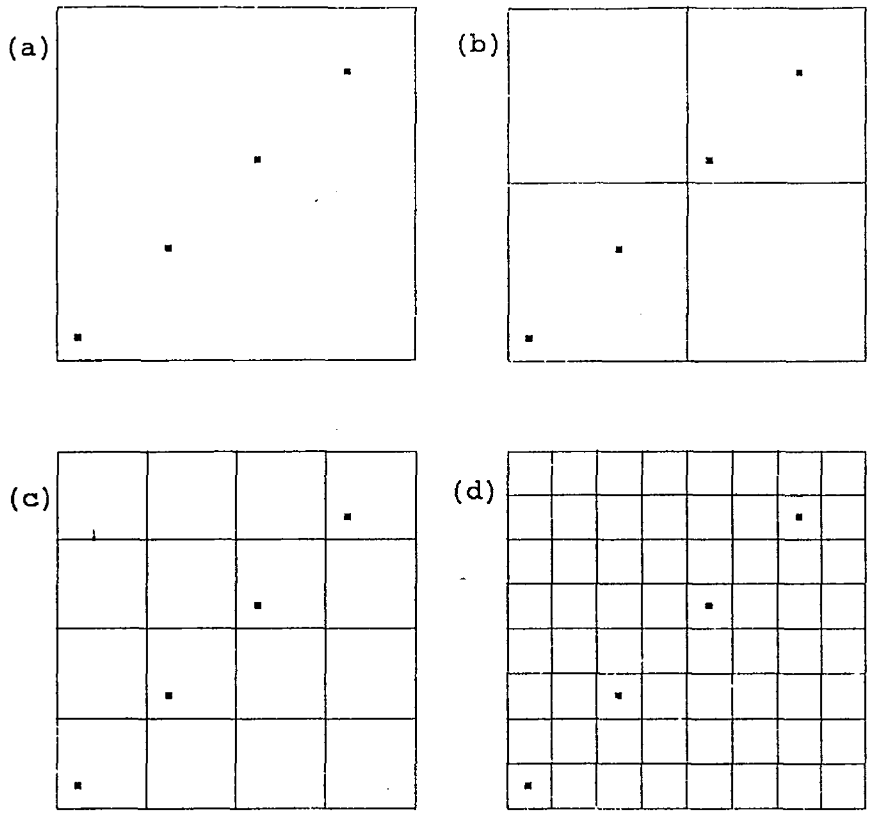
Determination of the grid size. Our algorithm for finding the grid size starts from some small initial sizes Δx and Δy and changes these sizes until a criterion for "density" ρ is satisfied.

Note that for the linear dependency $y = ax + b$ and evenly distributed points the "density" ρ becomes larger than 1 when $\Delta x > \Delta = X/N$ or $\Delta y > a\Delta = Y/N$, where N is the number of points, X is the range of x, and Y is the range of y. Based on these observations, we have developed the following algorithm to determine of the grid size in the case of unknown error. ρ_0 is the minimum required "density" of points in non-empty cells.

Algorithm: Determine grid size

```
    $\Delta x \leftarrow X/2N\rho_0$,   $\Delta y \leftarrow Y/2N\rho_0$
    $\rho \leftarrow N$ / (# of non-empty cells)
```

Figure 2: Comparison of "density" ρ for cells which contain data and true density: (a) sample points lying along a straight line $y = ax + b$; (b) grid with $\Delta x = \Delta/2$, $\rho = 1$, density $= 1/16$; (c) grid with $\Delta x = \Delta$, still $\rho = 1$ but density $= 1/4$; (d) grid with $\Delta x = 2\Delta$, $\rho = 2$ and density $= 1$. Note that parameter ρ stays the same until Δx becomes larger than Δ, or Δy larger than $a\Delta$; Δ is the smallest difference between x coordinates of points.



```
**if** $\rho \leq \rho_0$ **then**
   **repeat**
      $\Delta x \leftarrow 2\Delta x, \quad \Delta y \leftarrow 2\Delta y$
      $\rho \leftarrow N$ / (# of non-empty cells)
   **until** $\rho > \rho_0$
**else**
   **repeat**
      $\Delta x \leftarrow \Delta x/2, \quad \Delta y \leftarrow \Delta y/2$
      $\rho \leftarrow N$ / (# of non-empty cells)
   **until** $\rho < \rho_0$
   $\Delta x \leftarrow 2\Delta x, \quad \Delta y \leftarrow 2\Delta y$
**end if**
**end algorithm**
```

The initial values of Δx and Δy are chosen to be $X/2N\rho_0$ and $Y/2N\rho_0$, respectively, because for monotonic functions with more or less evenly distributed points the resulting density ρ would be close to ρ_0. The additional factor 1/2 was introduced to avoid decreasing the grid size in cases when initial ρ is only slightly larger than ρ_0. Decreasing the grid size is more costly than increasing it, because the latter operation can be performed on the existing grid from the previous step, while in the former case the density grid must be build from data. From the definition of ρ one can see that its value is never smaller than 1. If the grid size is too small, the value of ρ is about 1.

The initial values of Δx and Δy are chosen based on the discussion of a monotonic function with evenly distributed points. However, as our algorithm makes none of these assumptions, in many situations Δx and Δy would be then changed: either increased or decreased, until the resulting "density" parameter ρ gets close to the required value ρ_0.

When ρ becomes significantly greater than 1, say its value is about 2, it means that there is on average about two points per each non-empty cells. At that moment for most data one can analyze functionality based on that grid size, therefore a good default value for ρ_0 is around 2. However, when the distribution of data is very uneven, ρ_0 should be increased.

2.4 Functionality test

The input to the functionality test is a frequency table of data for a grid based on $b(X)$ and $b(Y)$, the binned values of x and y. The grid $b(X) \times b(Y)$ can be (1) built based on the known error, (2) determined by the above algorithm, or (3) obtained from the already binned data (in most databases the attributes are already binned). The following algorithm determines whether it is worthwhile to search for an equation that fits the data:

```
Algorithm: Test functional relationship between $x$ and $y$
  given the table of actual record counts
    AV ← average number of records per cell
    for each value in b(X)
      find all groups of adjacent cells with counts > AV
      if # of groups > $\alpha$ then
        return NO-FUNCTION
      end if
    end for
    if average # of groups > $\beta$ then
      return NO-FUNCTION
    else
      return FUNCTION
    end if
end algorithm
```

The algorithm is controlled by two modifiable parameters, α and β, which are measures of local (α) and global (β) uniqueness in y, that correspond to the number of values of y for the same value of x. The default values used by 49er are: $\alpha = 2$ (experimental data) or 3 (databases), $\beta \approx 1.5$. For $\alpha = 3$ the functionality test fails when for a value in $b(X)$ there is more than 3 adjacent groups of cells with the above average density of points. This higher value of α solves the problem of rare outliers: there could be up to 2 outliers (that is, cells having an above average number of points) provided it happens very rarely. However, many outliers or frequent discontinuities in y should fail the test, therefore the value of β should be much smaller and close to one. Note, that the test with both parameters set to 1 corresponds to the strict mathematical definition of the functionality, as discussed earlier. Presence of error, noise, and other data imperfections force values of α and β to be larger than 1.

The values of α and β should be increased for sparse data with strong background noise, while they should be reduced if we are looking for high-precision functional dependencies in experimental data. If the background noise is strong, fluctuations in noise could result in cells having above the average number of points coming from pure noise. In that case, considering cells with the above average density of points may not always be a cure for the noise. In such cases one should increase the values of α and β to allow more discontinuities in y, coming from noise. On the other extreme, if the data result from high-precision experiments with very little noise, discontinuities in y usually would reflect the true nature of the dependency in data (if any) and therefore should be treated more seriously: α and β should be decreased.

Generally, for high-precision data, α should be set to 2. Note, that this is practically the smallest value (α is integer). The test with $\alpha = 1$ frequently reports lack of functionality even for artificial data generated for a simple function plus very small error. On the other extreme, $\alpha = 4$ results in assigning functionality to most datasets, sometimes even when the data contains only noise (provided that β is increased, too). Therefore, α should be either 2 (high-precision experimental data) or 3 (most databases). For the similar reasons, the value of β should always be larger than 1. Its value can be estimated by N_{extra}/N_X, where N_X is the number of bins in X while N_{extra} is the total number of permitted groups of cells other than those following a (hypothetical) function. Again, for high-quality experimental data with very small noise and small error one could restrict N_{extra} to 1 or 2. However, for most databases the value of N_{extra} should be significantly larger, due to strong noise.

Although the test was originally designed for real-valued attributes, it can also be used for ordinal attributes provided that the number of different attribute values is not too small (in practice, there should be at least 5 values).

3 An application: discovery of functionality in 49er

49er is a machine discovery system that analyzes large databases in search for regularities hidden in those data. In the first phase of its exploration of a database, 49er looks for regularities holding for 2 attributes (variables). We will focus on that problem.

The two basic types of regularities considered by 49er are contingency tables and equations. While analysis of contingency tables is relatively fast (for example, 49er needs about 0.1s for a typical contingency-all regularity), the search for equations is quite costly (3-5 seconds for the same data). Therefore 49er uses domain knowledge about attributes (for example, to avoid search for equations for nominal attributes) and the test for functionality to avoid the search for equations when there is no hope to find them.

Table 1 presents empirical data which show advantages of the functionality test on several test runs. While the number of detected regularities was only slightly smaller when the test has been applied, the computation time was significantly shorter. In these tests some 26,000 of data subsets were examined by 49er. Only few of them can be reasonably approximated by functional dependencies. Most discovered equations are very weak. Equations "lost" when the test was used belong to the weakest of all discovered equations and their number varies when one slightly changes α and β. Table 2 shows one of the "lost" equations: note that one can hardly see any functional dependence for those data.

Table 1: Comparison of 49er execution times with and without test for functionality. Tests were conducted on about 1400 records, 10 attributes, and an average 10 values per attribute. 49er considered about 26,000 datasets in those data. The results show that run-time was reduced significantly while only about 10% of equations were not discovered.

Test number	Functionality test used?	Number of equations	CPU time	Comments
1	yes	71	258 minutes	database of
2	no	77	610 minutes	1400 records
3	yes	24	58 minutes	same data but
4	no	26	80 minutes	a more shallow search

Tests presented in Table 3 were designed to check also the overhead introduced by the functionality test: a strong functional dependence was present in all analyzed subsets of data. The execution times show that that overhead is negligible. Note that this time no equation was lost. Table 4 presents a strong equation that was discovered in one of the subsets: as opposed to data in the table 2, this time functional dependence is clearly visible.

The example data of table 4 also clearly shows why we had to modify the mathematical definition of functionality. First, although the discovered equation is quite strong, one can still easily see the presence of error: for many values of x there is more than one corresponding value of y. Second, the nature of this dependency (quite well approximated by a logarithmic function) is that for the first value of x most of y values are present.

Many tests conducted on real and artificially generated data show that only a small percent of equations that weakly fit data could be sometimes lost when the test is used. Our tests reported in Table 3 demonstrate a very small overhead introduced by the test.

Table 2: Actual frequency grid summarizing data for which a quite strong equation was found: $(CANX) = \log(a + b(HST))$. The fitted values of parameters a and b and their standard deviations σ_a and σ_b are $a = -0.23$, $\sigma_a = 0.13$, $b = 0.097$, $\sigma_b = 0.030$.

CANX

9	1	1	2	0	1	0	0	0	0
8	1	3	3	0	1	0	0	0	0
7	5	2	1	2	0	1	1	1	2
6	5	5	4	5	5	4	4	2	1
5	1	5	9	8	6	4	7	4	1
4	4	5	6	13	12	5	5	5	7
3	2	1	6	5	8	10	7	2	3
2	4	7	10	4	12	4	4	2	2
1	2	4	3	4	2	3	3	4	3
0	0	2	3	1	3	0	3	0	3
-1	2	0	0	2	3	1	1	1	1
-2	0	1	1	0	0	0	0	0	0
-3	0	0	1	0	0	0	0	0	0
	2	3	4	5	6	7	8	9	10

HST

Table 3: Comparison of 49er execution times with and without test for functionality. Here 49er focused only on pairs of attributes that had functional dependencies, known or discovered earlier, so that the test almost always resulted in the "yes" answer. Note that execution times for tests 5 and 6 are almost the same — it means that the overhead introduced by the test for functionality can be neglected compared to time saved by not using the Equation Finder.

Test number	Functionality test used?	Number of equations	CPU time	Comments
5	yes	23	699 seconds	6000 records, only for pairs of attributes with functional dependencies
6	no	23	695 seconds	

The functionality test can be also applied as a tool for discovering the *presence* of functional dependencies between attributes. When requested, 49er can use only this test while analyzing a database. The results are of interest to users investigating the existence of functional dependencies in data, but can be also used

Table 4: Actual frequency grid for a data subset where a strong equation was found: $(PAYGRADE) = \log(a+b(LOS))$. The fitted values of a and b and their standard deviations are $a = 3.13$, $\sigma_a = 0.03$, $b = 4.67$, $\sigma_b = 0.12$. Note, that the ratio of deviation to the corresponding parameter value is significantly smaller than for the previously discussed equation.

PAYGRADE

7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
6	0	0	0	0	0	3	3	12	21	17	11	7	10	8	4	1	0
5	4	2	2	69	146	68	44	34	24	11	4	3	2	2	0	0	0
4	10	12	60	102	90	31	20	8	5	3	2	0	0	0	0	0	0
3	602	309	173	72	43	14	10	4	6	4	1	1	1	0	0	1	0
2	1242	31	5	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	2371	1	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	20

LOS

to modify the default search strategy of 49er. The latter application is especially useful for databases with many attributes. Our functionality test can be quickly applied to many pairs of attributes, and the results of this preliminary search can be used to direct the search in the most promising directions, without considering combinations of attributes for which the functionality test failed or returned high values of β.

4 Discovery of multifunctions

We will now discuss an extension of the functionality test that leads to discovery of multifunctions. The data pass our functionality test when, for most values of x, there is only one continuous group of cells with counts above the average. However, if the test consistently shows two groups of cells for a subrange of x, it could mean that there are *two* functional dependencies in data: $y = f_1(x)$ and $y = f_2(x)$.

The existence of two or more functional dependencies in data is practically useful. For example, if one collects data on the weight of African elephants of different age, one can observe that there seems to be two distinct dependencies between weight and age. A closer look could show that there are actually two kinds of African elephants: large bush elephants (*Loxodonta africana*) and smaller forest elephants (*Loxodonta cyclotis*). Such a discovery of two or more functional dependencies in data could lead to the discovery of a classification. Piatetsky-Shapiro and Matheus (1991) present another method for detection of multifunctions. However, their approach is currently limited to linear functions and is $O(N^2)$.

We have implemented Multiple Equation Finder (called MEF) that uses an extended version of the functionality test to discover multiple functional relationships. If the test returns a high value of β, MEF checks continuity of cell groups in the x direction, then partitions the data and runs Equation Finder for every group of adjacent cells, and finally tries to extend range of discovered equations (if any) and reduce number of groups by merging them, if they can be described simultaneously by means of the same equation.

```
Algorithm: Multiple Equation Finder
   run Test for Functionality
   if average # of groups < $\beta$ then
      run Equation Finder
   else
      for each group of adjacent high-density cells
         merge the group with others if they are adjacent in $x$
      end for
      for every group spanning through many subranges of $x$
         run Equation Finder
      end for
      repeat
         merge two groups when they have similar equations or
               equation of one groups describes data in the other group
      until no merging happened in the last iteration
   end if
end algorithm
```

Detailed description of group merging goes beyond the scope of this article and will be presented elsewhere.

5 Conclusions

In this article we have presented a test which detects existence of functional dependencies in data. Currently, the functionality test is used (1) in the database mining system 49er in order to decide whether to invike the search for equations and (2) to find multiple equations in data.

In 49er, if the user is interested in discovery of equations, the functionality test is applied to each dataset. Based on the results of this quick test, 49er decides whether the computationally expensive search of Equation Finder should be performed or not. In our applications on various databases we have found that the functionality test significantly reduces the search time, thus increasing the overall performance of 49er.

Multiple Equation Finder uses the functionality test in a different way: the test determines whether and how the data should be partitioned so that a single equation can become a search target in each partition. The functionality test is only one of application tests used by 49er. New tests can be very easily added to the system. Generally, for each type of regularity a number of application tests can be defined.

Our test for functionality works very well for numeric or ordinal data.

When it is known that the data contain very small error and no noise and attributes have only few values, one could apply a simpler test, or a test for discrete-valued attributes used by Piatetsky-Shapiro (1992) in the Knowledge Discovery Workbench (Piatetsky-Shapiro and Matheus, 1991).

Acknowledgment: Special thanks to two anonymous reviewers for very helpful comments and suggestions.

References

Falkenhainer, B.C. & Michalski, R.S. (1986). Integrating quantitative and qualitative discovery: The ABACUS system. *Machine Learning*, Vol.1, pp.367-422.

Hoschka, P. & Klösgen, W. (1991). A Support System for Interpreting Statistical Data, in: Piatetsky-Shapiro G. & Frawley W. eds *Knowledge Discovery in Databases*, Menlo Park, Calif.: AAAI Press.

Klösgen, W. (1992). Patterns for Knowledge Discovery in Databases in: Żytkow J. ed *Proceedings of the ML-92 Workshop on Machine Discovery (MD-92)*, National Institute for Aviation Research, Wichita, Kansas, pp.1-10.

Langley, P., Simon, H. A., Bradshaw, G. L. & Żytkow, J. M. (1987) *Scientific discovery: Computational explorations of the creative processes.* Cambridge, MA: MIT Press.

Moulet, M. (1992) ARC.2: Linear Regression In ABACUS, in: Żytkow J. ed *Proceedings of the ML-92 Workshop on Machine Discovery (MD-92)*, National Institute for Aviation Research, Wichita, Kansas, pp.137-146.

Nordhausen, B., & Langley, P. (1990). An Integrated Approach to Empirical Discovery. in: J.Shrager & P. Langley (eds.) *Computational Models of Scientific Discovery and Theory Formation* (pp. 97-128), Morgan Kaufmann Publishers, San Mateo, CA.

Piatetsky-Shapiro, G., (1992) Probabilistic Data Dependencies. in: J.M. Żytkow (ed.) *Proc. of the ML-92 Workshop on Machine Discovery (MD-92)*, National Institute for Aviation Research, Wichita, KS.

Piatetsky-Shapiro, G. and C. Matheus, (1991) Knowledge Discovery Workbench, in: G. Piatetsky-Shapiro ed. *Proc. of AAAI-91 Workshop on Knowledge Discovery in Databases*, pp. 11–24

Zembowicz, R., Żytkow, J.M. (1992) Discovery of Equations: Experimental Evaluation of Convergence, *Proceedings of the Tenth National Conference on Artificial Intelligence*, The AAAI Press, 1992, 70-75.

Żytkow, J.M. (1987) Combining many searches in the FAHRENHEIT discovery system, *Proceedings of the Fourth International Workshop on Machine Learning.* Irvine, CA: Morgan Kaufmann, 281-287.

Żytkow, J., and Baker, J., (1991) Interactive Mining of Regularities in Databases. In *Knowledge Discovery in Databases*, eds. G. Piatetsky-Shapiro and W. Frawley. Menlo Park, Calif.: AAAI Press.

Żytkow, J.M. & Zembowicz, R. (1993) Database exploration in search of regularities, to be published in: *Journal of Intelligent Information Systems*, Vol.2, 1993.