Center
for
Economic Research

No. 2000-63

COMBINING DOMAIN KNOWLEDGE AND DATA IN DATAMINING SYSTEMS

By Hennie Daniels and Ad Feelders

July 2000

ISSN 0924-7815

Combining domain knowledge and data in datamining systems

Hennie Daniels[1,2] and Ad Feelders[1]

[1] Tilburg University
CentER for Economic Research
PO Box 90153
5000 LE Tilburg, The Netherlands
A.J.Feelders@kub.nl
[2] Erasmus University Rotterdam
ERIM Erasmus Research Institute for Management Studies
PO Box 1738
3000 DR Rotterdam, The Netherlands

1 Introduction

Despite the predominant attention for analysis in the datamining literature, data selection and preprocessing have a substantial influence on the success of data mining projects. Since a database is always an imperfect description of a real business process, there are numerous problems to overcome. If the description of the domain is too limited, essential patterns in the environment may not have a counterpart in the database. On the other hand, a lot of information stored in the database may be superfluous with respect to the problem under consideration. Furthermore, incorrect information such as missing attribute-value pairs and wrong attribute-values may lead to incorrect conclusions. Finally, retrieval, merging and aggegration of the data can be a major task on its own especially if the information is stored in distributed and heterogeneous databases.

Recently, it has been argued that the usual datamining algorithms may also have several shortcomings when used to support real-world decision-making [FDH00]. These limitations are due to a variety of factors such as:

- The incompatibility of domain knowledge with patterns in transaction databases, such as knowledge embedded in corporate policy rules and business regulations;

- Difficulties in representing uncertainty, imprecision or incompleteness of knowledge about the considered phenomena;
- The unsuitability in adequately representing a priori knowledge, such as causal knowledge;
- In some applications, computational expense.

Consequently, there is a growing interest in integrating the traditional datamining software, which derives knowledge purely from data alone with descriptive methods for encoding domain knowledge. It is well-known that a manager's assessment of the plausibility of knowledge derived from databases using datamining techniques, depends on his perspicacity and intuition. He will often also need to convince other managers about the correctness of this knowledge, but current datamining tools do not contribute much to this process of persuasion. The current limitations of datamining software would be partly removed if the explicativity of the system to decision makers were improved. One such improvement would be the incorporation of explicit knowledge based on experience and intuition of the decision maker or analyst in the datamining process. There is a great scope here for an integration of knowledge extracted from experts in the domain encoded in some accessible way, with knowledge derived from conventional datamining algorithms. There is a variety of applications where this could be useful like:

- Risk assessment in the presence of both qualitative knowledge and legal or contractual constraints.
- Knowledge structuring and concept identification in evaluation decision processes such as credit loan evalution, risk-assessment and fraud detection.
- Validation of business rules especially in a distributed user environment.
- All kinds of price models for trend analysis or automatic trading employed in combination with transaction databases.

Cases have been explored in the fields of reinsurance, marketing and credit control, and more are envisaged. In many administrative tasks there is a need to comply with existing legislation or business policy rules. The rules must be enforced in business processes which can be a problem if knowledge is derived with datamining

algorithms from distributed databases. In this paper we will show that datamining systems can be succesfully combined with explicit domain knowledge , yielding improvement of transparancy and effectiveness of the complete system.

We refer to such programs as Hybrid Datamining Systems (HDS), they are based on the expert domain knowledge or rules used in business operations coupled with datamining techniques that derive additional knowledge from cases stored in the database.

In learning from data one can imagine two extreme situations with respect to the availability of domain knowledge. The first is that no prior knowledge whatsoever is available, the second is that the relationship is known with certainty up to a limited number of parameters. Both extremes are unlikely to occur in practice. Data mining is often associated with the situation where little prior knowledge is available and an extensive search over possible models is performed.

The estimation of economic relationships from empirical data is studies in the field of econometrics. The textbook approach to econometrics assumes the other extreme with respect to a priori knowledge. In the model specification stage the relevant explanatory variables and the functional form of the relationship with the dependent variable are derived from economic theory. Then the relevant data are collected and the model is estimated and tested.

Applied econometrics does not conform to this textbook approach, but is often characterized by what Leamer ([Lea78]) calls *specification searches*. Researchers who perform a specification search on the data are actually accused of *data mining* in the econometrics literature ([Lea78,Lov83]). The problem is that standard econometric model testing is no longer valid in this case. In the data mining community this problem is well-known (overfitting), and out-of-sample testing and cross-validation have become standard practice.

Why does the applied econometrician not follow the textbook approach but perform an amount of specification search? One reason might be the wish to obtain "significant" results that warrant publication. The best model, selected from the large amount of models tried, is sometimes presented as if it where specified completely in advance and no other models were tried. The other reason is that

very often "economic theory" is not formulated in such a way that a unique model specification may be derived from it.

In data mining we usually start at the other end of the spectrum and assume very little prior knowledge is available. Of course one has to have some ideas, for how else does one decide which explanatory variables to include in the model? But often the algorithm is able to select the relevant variables from a large collection of variables (like in stepwise regression) and furthermore flexible functions are used, i.e. there is little known about the functional form of the relation between the dependent and explanatory variables. Even though data mining is often applied to domains where little theory is available, in some cases useful prior knowledge is available, and one would like the mining algorithm to make use of it one way or the other.

One form of prior knowledge that is often available (in economic theory) is about the sign of a relation between dependent and explanatory variable. Economic theory would state that people tend to buy less of a product if its price increases (ceteris paribus), so there would be a negative relationship between price and demand. The strength of this relationship and the precise functional form are however not always dictated by economic theory. The usual assumption that such relationships are linear are more for mathematical convenience than anything else.

2 Domain knowledge in data mining

The term *domain knowledge* (*background knowledge*, *prior knowledge*) are used for different types of knowledge in the data mining literature. We make a broad distinction between

1. Auxiliary knowledge.
2. Normative knowledge about the model to be constructed.
3. Knowledge about the *data generating process*.

An example of the first category concerns the cost of measurement of different variables, e.g. in the context of medical diagnosis (see [Nn91]). In that case one would like to consider both the amount of information and cost of measurement of a variable in model construction. Another form of auxiliary knowledge frequently encountered is knowledge about the hierarchical structure of the domains

of attributes [Nn91]. In figure 1 we give an example of such an hierarchy for an application concerning the analysis of traffic accidents. The objective of this study was to find profiles of *fatal* traffic accidents. One important attribute indicates which objects are involved in the accident. Although the attribute values are recorded at the lowest level of generalization (the leaves of the hierarchy), the data mining algorithm can use the hierarchy to find rules at a higher level of abstraction.

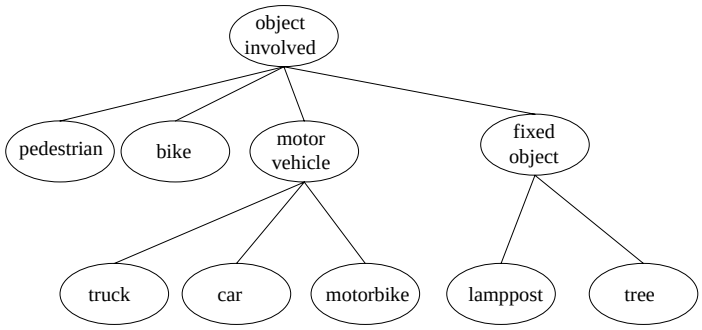

Fig. 1. Hierarchy of attribute values for traffic accident application

Normative knowledge may be important if the objective of data mining is to find a model that will be used in decision making, e.g. acceptance/rejection decisions. Examples are loan acceptance and employee selection. A common sense requirement is that the decision rule should be *monotone* with respect to certain variables. In loan acceptance the decision rule should be monotone with respect to income for example, i.e. it is not acceptable that an applicant with high income is rejected whereas another applicant with low in-

come and otherwise equal characteristics is accepted. Monotonicity of a relationship is a very common form of domain knowledge, and therefore we provide much attention to it in this study.

Finally, knowledge of the data generating process or the "true" model is also an important type of domain knowledge. In the next section we give an example of this type of prior knowledge for the domain of economics.

3 Economic domain knowledge in regression

According to most textbooks, econometrics concerns itself with the application of tools of statistical inference to the empirical measurement of relationships postulated by economic theory. Regression analysis is by far the most widely used technique in econometrics. This is quite natural since economic models are often expressed as (systems of) equations where one economic quantity is determined or explained by one or more other quantities.

The a priori domain knowledge is primarily used in the *model specification* phase of the analysis. Such a priori knowledge is supposed to be derived largely from *economic theory*.

Model specification consists of the following elements:

1. Choice of dependent and explanatory variables.
2. Specification of the functional form of the relation between dependent and explanatory variables.
3. Restrictions on parameter values.
4. Specification of the stochastic process.

In practice economic theory is rarely so detailed that it leads to a unique (unambiguous) model specification (or there may be many rival theories). Especially the usual *ceteris paribus* clauses of economic theory yield some choices to be made when it comes to the empirical estimation and testing of relationships.

In applied econometrics usually alternative specifications are tried, and the specification, estimation and testing steps are iterated a number of times (see [Lea78] for an excellent exposition of different types of "specification-searches" used in applied work).

As a historical note, this search for an adequate specification based on preliminary results has sometimes been called "data mining" within the econometrics community [Lea78,Lov83]. There is nothing wrong in principle with this behaviour, its combination however with classic testing procedures that do not take into account the amount of search performed have given "data mining" a negative connotation.

We give example from empirical demand theory to illustrate the different types of domain knowledge that may be used in the model specification phase. Empirical demand theory asserts that *ceteris paribus* an individual's purchases of some commodity depend on his income, the price of the commodity and the price of other commodities. We consider a simple demand equation taken from [Lea78]

$$\log D_i^o = a + b \log P_i^o + c \log Y_i + d \log P_i^g + \varepsilon_i$$

where D^o denotes the purchases of oranges, P^o the price of oranges, Y denotes income,P^g denotes the price of grapefruit, and index i stands for different households. The log-linear specification is chosen primarily for convenience. It allows us to interpret the estimated coefficients as elasticities, e.g. the estimate of b is interpreted as the price elasticity of demand, and the estimate of c as the income elasticity of demand. A priori we would expect that $b < 0$ (if price increases, *ceteris paribus* demand decreases). Likewise we would expect $c > 0$ and $d > 0$ (since grapefruits are a substitute for oranges).

According to (an) economic theory there should be absence of money illusion, i.e. if income and all prices are multiplied by the same constant, demand will not change. If we believe in the"absence of money illusion" we can add the restriction that the demand equation should be homogeneous of degree zero. For the log-linear specification this leads to the constraint $b + c + d = 0$.

Finally, for the stochastic part, one could assume that the random disturbances ε_i are normally distributed with mean 0 and constant variance, and ε_i independent of ε_j for $i \neq j$.

3.1 Monotonic regression

There are many economic regression and classification problems where it is known that the dependent variable has a distribution that is

monotonic with respect to its independent variables. A well-known example is labor wages as a function of age and education (see e.g. [MS94]) or in so-called hedonic price models where the price of a consumer good depends on a bundle of characteristics for which a valuation exists [HR78].

An example is treated in this paper where the price of a house is estimated as a function of the characteristics of the house. The traditional method used in isotonic regression is the *pool-adjacent violaters algorithm* [RWD88]. This method however only works in the one-dimensional case. A versatile non-parametric method is given in [MS94]. In general we have a dataset $(y^p, \mathbf{x}^p)$ of points in $\mathbb{R} \times \mathbb{R}^m$,which can be considered as a random sample of the joint distribution of $(y, \mathbf{x})$. The regression function we want to estimate is $E(y \mid \mathbf{x})$. We assume that $E(y \mid \mathbf{x})$ is monotonic increasing (more precisely non-decreasing). That is:

$$\mathbf{x}^1 \geq \mathbf{x}^2 \Rightarrow E(y \mid \mathbf{x}^1) \geq E(y \mid \mathbf{x}^2),$$

where $\mathbf{x}^1 \geq \mathbf{x}^2$ is defined as $\mathbf{x}^1_i \geq \mathbf{x}^2_i$ for all $i = 1, 2, \ldots, m$.

Now if t is any estimator of $E(y \mid \mathbf{x})$ we can make t into a monotonic regression by simply defining:

$$G^+(\mathbf{x}) = \max_{\mathbf{x}' \leq \mathbf{x}} t(\mathbf{x}'),$$

and

$$G^-(\mathbf{x}) = \min_{\mathbf{x}' \geq \mathbf{x}} t(\mathbf{x}').$$

Any convex combination of G^+ and G^- is a monotonic estimator of $E(y \mid \mathbf{x})$. G^+ is the smallest monotonic majorant of t and G^- is the largest monotonic minorant. For t one usually takes a kernel smoother with a variable bandwidth

$$t_a(\mathbf{x}) = \sum_k y_k \;\; K\left(\frac{\|\mathbf{x} - \mathbf{x}_k\|}{a}\right).$$

Noise in the data that may cause non-monotonic behaviour of the kernel smoother will be rectified by the maximization and minimization procedure in G^+ and G^- respectively. An undesirable side-effect of the method is that this type of noise is accumulated in G^+ and G^-.

4 Domain knowledge in trees

Tree-based algorithms such as CART and C4.5 are very popular in data mining. It is therefore no surprise that many variations on these basic algorithms have been constructed to allow for the inclusion of different types of domain knowledge such as the cost of measuring different attributes and misclassification costs. Another common form of domain knowledge concerns monotonicity of the allocation rule. Monotonicity of the allocation rule is often a common-sense requirement when the allocation rule is used for acceptance/rejection decisions like in loan evaluation. It would for example be hard to explain to an applicant why he is rejected whereas someone with a lower income but otherwise identical features is accepted.

First we make the notion of monotone classification precise. Let $\mathcal{X}$ be the feature space, with partial ordering $\geq$, and let $\mathcal{C}$ be a set of classes with linear ordering $\geq$. An allocation rule is a function $r : \mathcal{X} \to \mathcal{C}$ which assigns a class from $\mathcal{C}$ to every point in the feature space. Let $r(\mathbf{x}) = i$ denote that an entity with feature values $\mathbf{x}$ is assigned to the i^{th} class.

An allocation rule is monotone if

$$\mathbf{x}^1 \geq \mathbf{x}^2 \Rightarrow r(\mathbf{x}^1) \geq r(\mathbf{x}^2),$$

for all $\mathbf{x}^1, \mathbf{x}^2 \in \mathcal{X}$.

A classification tree partitions the feature space $\mathcal{X}$ into a number of hyperrectangles (corresponding to the leaf nodes of the tree) and elements in the same hyperrectangle are all asigned to the same class. As is shown in [Pot99], a classification tree is non-monotonic if and only if there exist leaf nodes t_1, t_2 such that

$$r(t_1) > r(t_2) \text{ and } \min(t_1) \leq \max(t_2),$$

where $\min(t)$ and $\max(t)$ denote the minimum and maximum element of t respectively.

In order to construct an allocation rule we have a dataset $(c^p, \mathbf{x}^p)$, which is called monotone if

$$\mathbf{x}^i \geq \mathbf{x}^j \Rightarrow c^i \geq c^j,$$

for all $i, j = 1, \ldots, p$.

Potharst [Pot99] provides a thorough study for the case that the training data may be assumed to be monotone. This requirement however makes the algorithms presented of limited use for data mining. For example, in loan evaluation the dataset used to learn the allocation rule would typically consist of loans accepted in the past together with the outcome of the loan (say, defaulted or not). It is very unlikely that this dataset would be monotone.

A more pragmatic approach is taken by Ben-David [BD95], who proposes a splitting rule that includes a measure of the degree of monotonicity of the tree in addition to the usual impurity measure.

To this end a $k \times k$ symmetric non-monotonicity matrix M is defined, where k equals the number of branches of the tree. The m_{ij} element of M equals 1 if branch i is non-monotonic with respect to branch j and 0 otherwise. Clearly, the diagonal elements of M are 0. Ben-David now defines a non-monotonicity index I as follows

$$I = \frac{W}{k^2 - k},$$

where W denotes the sum of M's entries, and $k^2 - k$ is the maximum possible value of W for any tree with k branches. Based on this non-monotonicity index the order-ambiguity-score of a decision tree is defined as follows

$$A = \begin{cases} 0 & \text{if } I = 0 \\ -(\log_2 I)^{-1} & \text{otherwise} \end{cases}$$

Finally the splitting rule is redefined to include the order- ambiguity-score

$$T = E + RA,$$

where T denotes the total-ambiguity-score to be minimized, E is the well-known entropy measure, and R is a parameter that expresses the importance of monotonicity relative to inductive accuracy.

A possible improvement of Ben-David's non-monotonicity index would to to weight the different branches according to their probability of occurrence. This makes sense because if two low-probability branches are non-monotonic with respect to each other, this is less severe than for two high-probability branches. The matrix M could now be adapted as follows. The m_{ij} element of M equals $p(i) \times p(j)$

if branch i is non-monotonic with respect to branch j and 0 otherwise, where $p(i)$ denotes the proportion of cases in branch i. The non-monotonicity index becomes

$$I = \frac{W}{(k^2 - k)/k^2} = \frac{W}{1 - 1/k},$$

where W is again the sum of M's entries, and the maximum is attained when all possible branches are non-monotonic with respect to each other and occur with equal probability $1/k$. W is an estimate of the probability that if we draw two points at random from the features space, these points turn out to lie in two leaves that are non-monotonic with respect to each other. The reader should note that $p(i) \times p(j)$ is only a crude approximation to the degree of non-monotonicity between node i and j because not all elements of i and j have to be non-monotonic with respect to eachother (some may be incomparable). We could estimate this proportion with bootstrapping, but then the algorithm would become computationally very complex. As a measure of the degree of non-monotonicity it is an improvement however over giving equal weight to each pair of non-monotonic branches.

To illustrate the idea we consider a simple example (see Table 1). This is a binary classification problem ($\mathcal{C} = \{0, 1\}$), whith two binary features x and y ($\mathcal{X} = \{(1,1), (1,2), (2,1), (2,2)\}$).

	$y=1$		$y=2$	
Class	0	1	0	1
$x=1$	14	136	20	0
$x=2$	66	34	70	0
Total	80	170	90	0

Table 1. Table with example data

We consider two possible splits from the root node where we have 170 cases from class 0 and 170 cases from class 1 (see Figure 2).

The tree on the left (r_1) sends half of the cases ($x = 1$) to the left subtree, and the other half ($x = 2$) to the right. We assume the allocation rule assigns a case in a node to the majority class, which is class 1 for the left subnode, and class 0 for the right subnode. Thus,

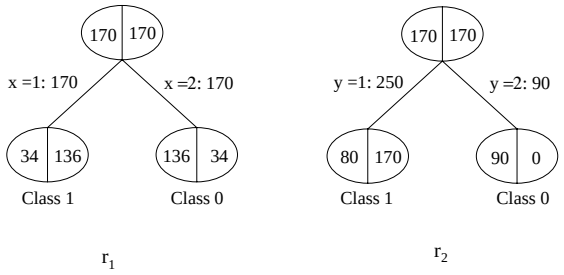

Fig. 2. Trees with the same impurity reduction but different degree of monotonicity.

the allocation rule is non-monotonic according to our definition. We compute the impurity reduction of this split with the Gini-index, i.e.

$$\Delta i(s,t) = i(t) - p_R i(t_R) - p_L i(t_L),$$

where

$$i(t) = \sum_{j \neq k} p(j|t)p(k|t)$$

is the Gini-index, and $p(j|t)$ is the relative frequency of class j at node t. For r_1 this is

$$(1/2 \times 1/2) - 1/2(1/5 \times 4/5) - 1/2(4/5 \times 1/5) = 9/100$$

The rightmost tree (r_2) splits on y, also leading to a non-monotonic allocation rule. We compute the impurity reduction of this split

$$(1/2 \times 1/2) - 25/34(8/25 \times 17/25) - 9/34(1 \times 0) = 9/100.$$

Thus, both splits give the same impurity reduction. Furthermore both splits are non-monotonic, but r_1 has a higher degree of non-monotonicity than r_2. If we were to draw two points at random

from the data, the probability that the class-allocations are non-monotonic (with respect to the variable actually used in the tree) is 0.5 for r_1, but only about 0.39 for r_2. One may also consider the degree of non-monotonicity on the entire (x, y) space. In that case the trees may still have different degrees of non-monotonicity. The allocation rules are summarized in table 2.

(x,y)	r_1	r_2	$p(x,y)$
(1,1)	1	1	15/34
(1,2)	1	0	2/34
(2,1)	0	1	10/34
(2,2)	0	0	7/34

Table 2. Allocation rules on (x, y) space with probabilities

The non-monotonicity matrices corresponding to these allocation rules are (all entries should be divided by 1156):

$$M_1 = \begin{pmatrix} 0 & 0 & 150 & 105 \\ 0 & 0 & 0 & 14 \\ 150 & 0 & 0 & 0 \\ 105 & 14 & 0 & 0 \end{pmatrix} \qquad M_2 = \begin{pmatrix} 0 & 30 & 0 & 105 \\ 30 & 0 & 0 & 0 \\ 0 & 0 & 0 & 70 \\ 105 & 0 & 70 & 0 \end{pmatrix}$$

Allocation rule r_1 has $W_1 = 538/1156 \approx 0.47$, and r_2 has $W_2 = 410/1156 \approx 0.35$.

Thus, whether we consider the degree of monotonicity on the entire space, or only on the space restricted to the variables actually used in the allocation rule, in both cases r_1 is less monotonic than r_2. This difference in degree of monotonicity is however not detected by the measure used in [BD95] since it just counts the number of pairs of nonmonotonic branches without weighting the for the probability of occurrence.

Of course one may argue that in this example, if both trees are split further they result in the same allocation rule. However, one has to consider the complex interaction of the splitting rule with other tree-growing rules such as stopping rules and pruning methods. This may result in the trees not being split any further, or being pruned back to the trees depicted in Figure 2.

To conclude this section we note that this approach is easily extended to the case where the allocation rule has to be monotonic only with respect to some features but not others (e.g. some features are not ordinal). We would only have to check monotonicity with respect to these designated variables.

5 Weighted rule systems

In most expert systems knowledge is stored in so called production rules. The rules correspond to chunks of articulated expert knowledge. In expert systems the rules interact with the user in a reasoning process and may reach conclusions just like experts can do in the domain of expertise. The usual standarized form in which the rules are encoded is the CNF (conjunctive normal form) syntax. Each rule consists of a IF and THEN part and the IF part is denoted in CNF syntax:

R_i: **IF** (A or B or ...) and (C or D or....) and ... **THEN** RHS

Here A,B,C,D are predicates that contain variables of the domain and RHS is the right hand side of the rule which stands for a conclusion of the rule. If the system operates in a forward reasoning mode new conclusions are generated during the consultation until the user is satisfied. Below we will show how production rules of this form can be integrated in a general framework with data. We developed this approach during a project on fraud detection in life insurance. The method is described here in a symplified way, but can be applied to a wide range of applications. Each rule corresponds to a risk indicator articulated by an expert. Suppose the rules are numbered R_1, R_2, R_3, etc. The rules can only yield the result true or false when applied to a certain case. The more rules apply, the larger the risk of fraud. The expert however does not know how the risk indicators can be combined to obtain a final risk score. In the case study at hand however there also exists a database with cases of proven fraud. The estimated probability of fraud is now written as the weighted sum of the invidual indicators, normalized such that the outcome is a number between 0 and 1:

$$p = \frac{\exp(\sum w_i r_i)}{1 + \exp(\sum w_i r_i)},$$

where $r_i = 1$ if rule R_i evaluates to true and 0 otherwise. The weights are real numbers.

The optimal weights can be found by fine tuning the weights using a regression on the database. This can be done by using a standard logistic regression model with binary predictors (see for example [Rip96], chapter 3, section 5). The weighted rules approach is a very flexible framework for combining declarative knowledge in combination with patterns.

6 Domain knowledge in neural networks

6.1 Monotonic neural networks

It is well-known that neural networks can be used to build flexible estimators. It can be shown that any relation can be approximated by a neural network with one hidden layer and sufficiently many neurons. The process of controlling the flexibility of neural networks in practical regression or classification problems is often cumbersome. Especially when the number of hidden neurons is large, neural networks have a tendency to overfit the data. This may lead to bad out of sample performance. Various approaches have been suggested to cope with the overfitting problem. The regularisation of the network can be done using domain independent methods such as weight decay, cross-validation and Bayesian approaches ([Rip96], chapter 4, section 3). In this paper we focus on methods that are based on the incorporation of domain knowledge. In this way one can decrease the variance of the network without increasing bias. Since the error term of the neural network approximation can be written as the sum of a bias and variance term, this will also diminish the total error. (see e.g.[DK99]) The method proposed can be succesfully combined with other regularisation methods. As mentioned in section many classification problems in economics and accounting possess monotonicity properties. The implementation of monotonicity constraints in neural networks can be done in different ways. In [Wan94] the monotonicity of the neural network is guaranteed by enforcing constraints on the weights during the training process. Here we apply a class of neural network that are monotonic by construction.This class is obtained by considering multilayer neural networks with non-

negative weights. In can be shown that the elements of this class can approximate any monotonic increasing function.

For a neural network with one hidden layer and one output neuron we have

$$y = \sum_{i=1}^{h} v_i \; f(\sum_{j=1}^{n} w_{ji} x_j + \theta_i)$$

where v_i denotes the weight connecting hidden neuron i with the output, f is the squashing function, w_{ji} is the weight connecting input j with hidden neuron i, and θ_i is the threshold of hidden neuron i. It can be easily seen that y is monotonic increasing (non-decreasing) if v_i and w_{ji} are positive (non-negative).

The maximum number of layers required is theoretically equal to the number of inputs but in many case studies it turned out that less will do. The training algorithm for monotonic neural networks that we have developed is a modification of the standard backpropagation algorithm. There are two ways of enforcing positive weights. The first one is by adding a bias term to the error function of the neural network such that the negative weights are penalised. The weight of the penalty term is gradually increased which eventually leads to a solution with only positive weights, corresponding to a monotonic network.In the second approach negative weights are set to zero in each step of the modified backpropagation algorithm. Both algorithms have been extensively studied on artificially generated datasets. The performance differs slightly but not significant. Testing on artificially generated datasets is preferable to adjust and finetune the algorithms, since everything is under control. In this paper we do not report on these studies but prefer to illustrate the method with a case study from real world.

7 Den Bosch Housing Data

In this case study we study a neural network predicting the price of a house in the city of Den Bosch. It is a medium sized Dutch city with approximately 120,000 inhabitants. The basic principle of a hedonic price model is that the consumption good is regarded as a bundle of characteristics for which a valuation exists ([HR78]). The price of

the good is determined by a combination of these valuations:

$$P = P(x_1, \ldots, x_n)$$

In the case at hand the variables $x_1, x_2, ., x_n$ are characteristics of the house.The explanatory variables have been selected on the basis of interviews with experts of local house brokers, and advertisements offering real estate in local magazines. The most important variables are listed in table 3.

Symbol	Definition
DISTR	type of district, four categories ranked from bad to good
SURF	total area including garden
RM	number of bedrooms
TYPE	1. apartment
	2. row house
	3. corner house
	4. semidetached house
	5. detached house
	6. villa
VOL	volume of the house
GARD	type of garden, four categories ranked from bad to good
GARG	1. no garage
	2. normal garage
	3. large garage

Table 3. Definition of model variables

In the simulation study we compare ordinary neural networks and monotonic neural networks. Firstly a ordinary neural network is trained on the dataset using 5 fold cross-validation. The error R^2 varies between 0.8089 and 0.9288. As a check for monotonicity we computed the monotonicity indices for each of the variables. All indices are approximately equal to 1 with accuracy 0.5×10^{-3}. Except for the variable GARG, which is relatively unimportant compared to the others (see table 4).

This is in line with economic intuition. In the next step we trained ordinary neural network s and monotonic neural networks with different number of hidden neurons up to 20 in the hidden layer. The results of the experiments with ordinary neural networks and monotonic networks are listed in table 5.

Variable	mon(x_i)	Sign
DISTR	1.00	+
SURF	1.00	+
RM	1.00	+
TYPE	1.00	+
VOL	1.00	+
GARD	1.00	+
GARG	0.56	+

Table 4. Monotonicity indices of the Den Bosch house price model

	Normal Neural Network					
# hidden n.	5	10	15	15	20	20
learning rate	0.1	0.1	0.1	0.1	0.1	0.01
momentum	0.9	0.9	0.1	0.9	0.9	0.1
R^2 (train)	0.9125	0.9423	0.9192	0.9748	0.9750	0.9517
R^2 (test)	0.8973	0.8207	0.8174	0.7366	0.6716	0.6926

	Monotonic Neural Network					
# hidden n.	5	10	15	15	20	20
learning rate	0.1	0.1	0.01	0.01	0.1	0.1
momentum	0.9	0.9	0.9	0.01	0.1	0.9
R^2 (train)	0.8679	0.8535	0.8365	0.8646	0.8491	0.8612
R^2 (test)	0.8815	0.9096	0.9239	0.9055	0.9167	0.9085

Table 5. Performance of normal and monontonic networks

It is clear from the table that monotonic neural networks show better out-of-sample performance and smaller variations of R^2 on the training set and test set.

8 Conclusion

The goal of dataming is to derive valuable business knowledge from patterns in databases. In the majority of cases there is theoretical and domain dependent knowledge available to the decision maker beforehand. In this paper we have shown that the effectiveness of datamining systems can be substantially be improved by using expert knowledge. We explicitly studied the framework of combining knowledge and data for different datamining algorithms, with a focus on decision trees, regression trees and neural networks. Those being the most important in economic decision making. We also showed that the framework developed may serve as a tool to implement normative requirements, that are often enforced by third parties in a business setting. Futhermore we introduced a versatile model for integrating rule based systems with statistical regression techniques, often necessary when decisions have to be taken in agreement with legislation or business policy rules. The metods are illustrated in practical case studies: predicting house prices and the generation of profiles of fatal traffic accidents.

References

[BD95] A. Ben-David. Monotonicity maintenance in information-theoretic machine learning algorithms. *Machine Learning*, 19:29–43, 1995.

[DK99] H. Daniels and B. Kamp. Application of MLP networks to bond rating and house pricing. *Neural Computation and Applications*, 8:226–234, 1999.

[FDH00] A. Feelders, H. Daniels, and M. Holsheimer. Methodological and practical aspects of data mining. *Information & Management*, 2000. to appear.

[HR78] O. Harrison and D. Rubinfeld. Hedonic prices and the demand for clean air. *Journal of Environmental Economics and Management*, 53:81–102, 1978.

[Lea78] E. Leamer. *Specification Searches: Ad Hoc Inference with Nonexperimental Data*. Wiley, 1978.

[Lov83] Michael C. Lovell. Data mining. *The Review of Economics and Statistics*, 65(1):1–12, 1983.

[MS94] H. Mukarjee and S. Stern. Feasible nonparametric esimation of multiargument monotone functions. *Journal of the American Statistical Association*, 89(425):77–80, 1994.

[Nn91] M. Núñez. The use of background knowledge in decision tree induction. *Machine Learning*, 6:231–250, 1991.

[Pot99] R. Potharst. *Classification using decision trees and neural nets.* PhD thesis, Erasmus Universiteit Rotterdam, 1999. SIKS Dissertation Series No. 99-2.

[Rip96] B.D. Ripley. *Pattern Recognition and Neural Networks.* Cambridge University Press, Cambridge, 1996.

[RWD88] T. Robertson, F. Wright, and R.L. Dykstra. *Order Restricted Statistical Inference.* Wiley, 1988.

[Wan94] S. Wang. A neural network method of density estimation for univariate unimodal data. *Neural Computation & Applications*, 2:160–167, 1994.