# Integrating Economic Knowledge in Data Mining Algorithms

Daniëls, H.A.M.; Feelders, A.J.

*Publication date:*
2001


No. 2001-63

# INTEGRATING ECONOMIC KNOWLEDGE IN DATA MINING ALGORITHMS

By Hennie Daniels and Ad Feelders

September 2001

ISSN 0924-7815

Discussion paper

**Integrating economic knowledge in data mining algorithms.**

**Hennie Daniels [1,2,*] and Ad Feelders[3]**

[1]Tilburg University, CentER for Economic Research,Tilburg, PO Box 90153,
5000 LE The Netherlands, phone: +31 13 466 2026, fax: +31 13 466 3377, e-mail: daniels@kub.nl
[2]Erasmus University Rotterdam, ERIM Institute of Advanced Management Studies,
Rotterdam, The Netherlands.
[3] Utrecht University, Institute of Information and Computing Sciences, PO Box 80.089,
3508 TB Utrecht, The Netherlands, phone: +31 30 253 3176, fax: +31 30 251 3791,
e-mail: ad@cs.uu.nl

**Abstract.**

The assessment of knowledge derived from databases depends on many factors. Decision makers often need to convince others about the correctness and effectiveness of knowledge induced from data. The current data mining techniques do not contribute much to this process of persuasion. Part of this limitation can be removed by integrating knowledge from experts in the field, encoded in some accessible way, with knowledge derived form patterns stored in the database. In this paper we will in particular discuss methods for implementing monotonicity constraints in economic decision problems. This prior knowledge is combined with data mining algorithms based on decision trees and neural networks. The method is illustrated in a hedonic price model.

**Keywords**: data mining, prior knowledge, monotonicity, neural networks, decision trees.

[*] corresponding author. Tilburg University, CentER for Economic Research,Tilburg, PO Box 90153, 5000 LE The Netherlands, phone: +31 13 466 2026, fax: +31 13 466 3377, e-mail: daniels@kub.nl

## 1. Introduction.

The goal of a data mining system is to derive useful knowledge that is implicitly present in large company databases. In recent years there has been a lot of interest in theory, software and applications in virtually all business areas, where data are recorded [5,7,17]. Here a data mining system is to be understood as the complete system: the database or data-warehouse, software for mining and analyses, the knowledge derived from it and the part of the system supporting final decision making in a business setting. Despite the predominant attention for analysis in the data mining literature, there are more aspects that determine the performance of data mining systems in a business environment. For example data selection and pre-processing have a substantial influence on the success of data mining projects. 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. Finally, retrieval, merging and aggregation of the data can be a major task on its own especially if the information is stored in distributed and heterogeneous databases [6, 7].

Apart from the limitations concerned with data quality one encounters difficulties in the application of the model if the knowledge discovery process is conducted by a blind search. Frequent occurring causes are:

- Incompatibility of the model derived form transaction databases with knowledge embedded in corporate policy rules and business regulations. 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 data mining algorithms from distributed databases.
- Lack of interpretability of the model. Human decision makers require that the final model is easy to understand and in general do not accept black-box models. Quite often it is more important to gain insight in the decision problem, than to have accurate predictions.
- Knowledge representation at the wrong level of detail. Data mining algorithms often yield structures or models that are intractable for human decision makers due to their huge complexity.

Consequently, there is a growing interest in integrating the traditional data mining software, which derives knowledge purely from data alone with descriptive methods for encoding domain knowledge or meta-knowledge guiding the search process. There is a great scope here for integration of knowledge based on experience and intuition of domain experts (or knowledge from other sources) encoded in some accessible way, with knowledge derived from conventional data mining algorithms [17]. In a variety of applications this will improve the efficiency of data mining systems:

- Risk assessment in the presence of both qualitative knowledge and legal or contractual constraints.
- Classification and description of customer groups in evaluation decision processes such as credit loan evaluation, risk-assessment and fraud detection.
- Validation of business rules especially in distributed user environments.
- All kinds of price models for trend analysis or automatic trading employed in combination with transaction databases.

The integration of knowledge from different sources in data mining systems is a complex knowledge management problem and not fully understood yet. Cases have been explored in the fields of insurance [6], marketing [17] and bankruptcy prediction [12], and more are envisaged. In this paper we will show that data mining systems can be successfully combined with economic domain knowledge, yielding improvement of transparency and effectiveness.

In theory there are two extreme situations that may occur with respect to the availability of domain knowledge. The first is that no prior knowledge whatsoever is available, in contrast to the case where 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 cases close the first situation: little prior knowledge is available and an extensive search over possible models is performed. In econometrics one mostly 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. So in fact almost everything is known up to a limited number of parameters. Then the relevant data are collected and the model is estimated and tested. Applied econometrics however does not exactly conform to this textbook approach but is often characterised as specification searches (c.f.[9]). 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. 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 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.

In this paper we position ourselves somewhere between these extremes. We will identify various classes of prior knowledge that are important in data mining problems. We will focus on the implementation of a special form of a priori knowledge that is typical in economic decision problems, namely the sign of the relationship between the dependent and the explanatory variables. Economic theory would state for example 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. In section 3 more examples are listed and a precise definition of monotonicity is formulated. In section 4 and 5 a formal framework for the implementation of monotonicity in the most common data mining algorithms is derived. The results are illustrated in a so-called hedonic price model.

## 2. Types of knowledge.

The terms domain knowledge, background knowledge, and prior knowledge are commonly used to denote different types of knowledge in the data mining literature. We make a broad distinction between:

- Normative knowledge about the model to be constructed.
- Knowledge about the data generating process.
- Knowledge improving cost and search efficiency.

Normative knowledge may be important if the objective of data mining is to find a model that will be used in decision making, for example in acceptance/rejection decisions. Well known are cases in

loan acceptance and employee selection. A common sense requirement is that the decision rule should be monotonic with respect to certain variables. In loan acceptance the decision rule should be monotone with respect to income for example, because it is not acceptable that an applicant with high income is rejected, whereas another applicant with low income and otherwise equal characteristics is accepted. Monotonicity of a relationship is a very common form of domain knowledge, and therefore we elaborate further on this aspect in sections 3, 4 and 5.

Knowledge of the data generating process, which is also called data expertise, is also an important type of domain knowledge. Data expertise is required to explain strange patterns and remove pollution for example caused by data conversion or merging of databases. For example in a case where a large insurance company took over a small competitor, the insurance policy databases were joined. The start-date of the policies of the small company were set equal to the conversion date, because only the most recent mutation date was recorded by the small company. Without this knowledge of conversion, one might believe that there was an enormous "sales peak" in the year of conversion ([6]).

An example of the last category concerns the trade-off between the cost of measurement of variables and the gain of information. Such cases occur frequently in the context of medical diagnosis (c.f.[11]). In that case one would like to consider both the amount of information and cost of measurement of a variable in model construction. Knowledge about the hierarchical structure of the domain can often be applied to increase the efficiency of the search process and to improve the transparency of the model.

## 3. Monotonicity in economic decision problems.

In many economic regression and classification problems it is known that the dependent variable has a distribution that is monotonic with respect to the independent variables. 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. Another well-known example is the dependence of labour wages as a function of age and education ([10]). In many acceptance decisions rules should be monotone with respect to several variables as explained in section 2. Monotonicity is also imposed in so-called hedonic price models where the price of a consumer good depends on a bundle of characteristics for which a valuation exists ([8]). The number of examples is manifold.

The mathematical formulation of the monotonicity rule is straightforward. We assume that $y$ is the dependent variable and takes values in $\boldsymbol{Y}$ and the vector of independent variables is $x$ and takes values in $\boldsymbol{X}$ . In the applications discussed here, $\boldsymbol{Y}$ is a one-dimensional vector of prices or classes and $\boldsymbol{X}$ is a n-dimensional vector of characteristics of products or customers for example. Furthermore we assume that we have a data set $(y^p, x^p)$ of points in $\boldsymbol{Y}*\boldsymbol{X}$, which can be considered as a random sample of the joint distribution of $(y, x)$. In a regression problem we want to estimate $E(y \mid x)$. The monotonicity constraint for $E(y \mid x)$ can be formulated as:

$$x^1 \geq x^2 \Rightarrow E(y \mid x^1) \geq E(y \mid x^2), \qquad (3.1)$$

where $x^1 \geq x^2$ is a partial ordering on $\boldsymbol{X}$ defined by $x_i^1 \geq x_i^2$ for i = 1, 2, …, n.

In cases where we are dealing with a classification problem we have an classification rule r($x$) that assigns a class to each vector $x$ in $\boldsymbol{X}$. Monotonicity of r is defined by:

$$x^1 \geq x^2 \Rightarrow r(x^1) \geq r(x^2). \qquad (3.2)$$

(We assume that the label set is ordered.)
In the next section we discuss the implementation of monotonicity is tree-based classification algorithms. As an example we treat a hedonic price model where the price of a house depends the on the characteristics of the house. In section 5 we implement monotonicity in regression and neural networks models. The same house-pricing model is used for testing.

**4. Monotonic decision trees.**

Tree-based algorithms such as CART ([2]) and C4.5 ([17]) are very popular in data mining [12]. It is therefore not surprising 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. Here we will discuss the incorporation of monotonic classification rules in the sense of (3.2). A classification tree partitions the feature space $\boldsymbol{X}$ into a number of hyperrectangles, corresponding to the leaf nodes of the tree and elements in the same hyperrectangle are all assigned to the same class. As is shown in [13], 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.
Figure 1 shows an example of a pair of non-monotonic leaf nodes.

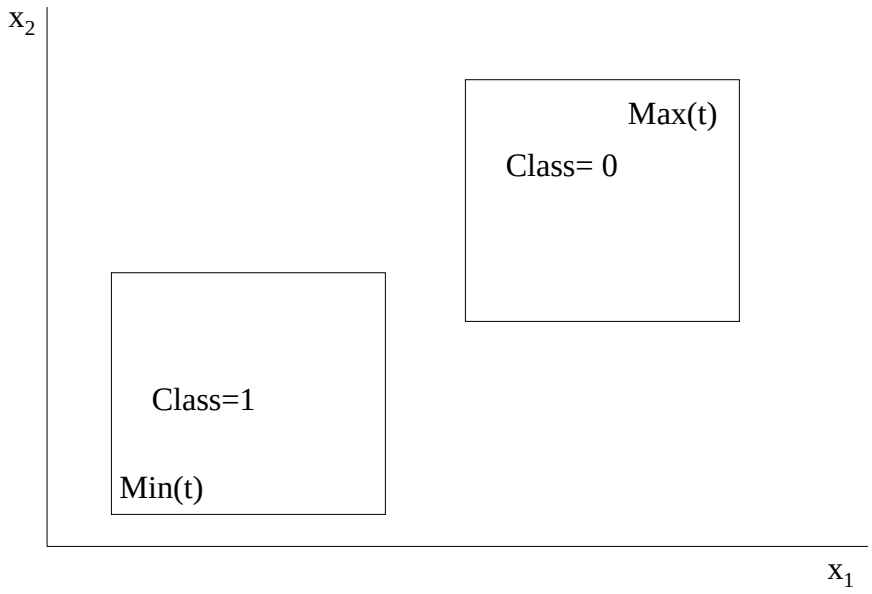


Figure 1. Non-monotonic leaf nodes.

A dataset $(y^p, x^p)$ is called monotonic if $x^1 \geq x^2 \Rightarrow y^1 \geq y^2$ , for all possible combinations

of data points $x^1 \geq x^2$ . Potharst ([13]) provides a thorough study for the case of monotonic training data. This requirement however reduces the applicability of the algorithms. For example, in loan evaluation the dataset 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 data set would be monotonic. A more pragmatic approach is taken by Ben-David [1], who proposes a splitting rule that includes a non-monotonicity index in addition to the usual impurity measure. To this end a k * k symmetric non-monotonicity matrix M is defined, where k equals the number of leaves of the tree constructed so far. The $m_{ij}$ element of M equals 1 if leaf $t_i$ is non-monotonic with respect to leaf $t_j$ and 0 otherwise. Clearly, the diagonal elements of M are 0.

A non-monotonicity index I is defined as follows: $I = W/(k^2 - k)$, 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 leaves [1]. Based on this non-monotonicity index the order-ambiguity-score of a classification tree is defined as follows

$A = 0$ if I=0,
$A = -(\log_2 I)^{-1}$ otherwise.

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

$$T = E + R\,A,$$

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.

The non-monotonicity index I gives equal weight to each pair of non-monotonic leaf nodes. A possible improvement of this index would be to give a pair $t_1$, $t_2$ of non-monotonic leaf nodes weight $p(t_1) * p(t_2)$ , where $p(t_i)$ denotes the proportion of cases in leaf $t_i$. The idea behind this is that when two low-probability leaves are non-monotonic with respect to each other, this violates the monotonicity of the tree to a lesser extent than two high-probability leaves. The reader should note that $p(t_1) * p(t_2)$ is an upper bound for the degree of non-monotonicity between node $t_1$and $t_2$ because not all their elements have to be non-monotonic with respect to each other. The matrix M could now be adapted as follows. The $m_{ij}$ element of M equals $p(t_i) * p(t_j)$ if leaf $t_i$ is non-monotonic with respect to leaf $t_j$ and 0 otherwise. The non-monotonicity index becomes

$$I = W/((k^2-k)/k^2) = W/(1-1/k), \qquad (4.1)$$

where W is again the sum of M's entries, and the maximum is attained when all possible leaves 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.

Using index (4.1) to determine the best split has certain drawbacks however. Monotonicity is a global property, i.e. it involves a relation between different leaf nodes of a tree. If the degree of monotonicity is measured for each possible split during tree construction, the order in which nodes are expanded becomes important. For example, a depth-first search strategy will generally lead to a different tree then a breadth-first search. Also, a non-monotonic tree may become monotonic after additional splits. Therefore we consider an alternative, computationally more intensive, approach in this study. Rather than enforcing monotonicity during tree construction, we generate many different

trees and check if they are monotonic. The collection of trees may be obtained by drawing bootstrap samples from the training data, or making different random partitions of the data in a training and test set. This approach allows the use of a standard tree algorithm except that the minimum and maximum elements of the nodes have to be recorded during tree construction, in order to be able to check whether the final tree is monotone. Furthermore, it has the additional advantage that one can estimate to what extent the assumption of monotonicity is correct.

For completeness a short description of a CART-like tree building algorithm is given. We assume that the domains of the independent variables, $x_1,\ldots,x_n$ are all totally ordered, and that the dependent variable $y$ takes its values in 1,2,…g, where g is the number of classes. The tree-building algorithm essentially works as follows. The root node of the tree contains all data points of the training sample. Then we compute the best split of type $x_j < c$ that partitions the training sample into data points with $x_j < c$ (the left child of the root node) and data points with $x_j \geq c$ (the right child of the root node). The best split in node t is defined as the split s that leads to the greatest reduction of impurity i:

$$\Delta\, i(s,t) = i(t) - P_L\, i(t_L) - P_R\, i(t_R), \qquad (4.2)$$

where $P_L$ denotes the proportion of cases in t going to the left child ($t_L$)and $P_R$ the proportion going to the right child ($t_R$). The impurity measure (4.2) indicates to what extent the data points in a node are of the same class. Common impurity measures are entropy and the gini-index:

$$i(t) = \sum_{k \neq l} p_k(t)\, p_l(t), \qquad k,l = 1,\ldots,g$$

where $p_k(t)$ is the proportion of the cases in node t that has class k. When all data points in a node have the same class, the gini-index reaches its minimum of 0, and when all g classes are equally represented the gini-index reaches its maximum of 1-1/g.

The training sample is partitioned into two sub-samples on the basis of the best split, and the algorithm is applied recursively to the sub-samples (hence the name recursive partitioning). A node is not split any further when it has impurity 0 (all data points have the same class), or some other stopping criterion applies, e.g. the node has too few data points left.

Below we give a description of the modification of the standard tree algorithm required to check the monotonicity of a tree.

**Algorithm 1: Maximum and minimum elements of the nodes.**

max(t,i) denotes the maximum of node t for variable $x_i$
min(t,i) denotes the minimum of node t for variable $x_i$

1. *Initialisation*: Let $t_0$ be the root node of the tree. Set $\max(t_0,i) = \text{Inf}$ and $\min(t_0,i) = -\text{Inf}$, for i = 1,…,n.
2. For arbitrary node t, let the best split be on variable $x_j$ with split value c. The left child $t_L$ contains the cases with $x_j < c$ and the right child $t_R$ the cases with $x_j \geq c$.
3. *Update left*: assign the max and min elements of $t_L$ as follows:
   $\min(t_L,i) := \min(t,i)$ for i = 1,…,n.

$\max(t_L,i) := \max(t,i)$ for $i \neq j$.
$\max(t_L,j) := c$

4. *Update right*: assign the max and min elements of $t_R$ as follows:
$\min(t_R,i) := \min(t,i)$ for $i \neq j$.
$\min(t_R,j) := c$
$\max(t_R,i) := \max(t,i)$ for $i = 1,\ldots,n$.

**Algorithm 2: Degree of non-monotonicity of a tree T.**

1. Determine all pairs $(t_1,t_2)$ of leaf nodes of T with $r(t_1) > r(t_2)$.
2. For each such pair, if $\max(t_2,i) > \min(t_1,i)$ for all $i = 1,\ldots,n$, then increment the degree of non-monotonicity with $2(p(t_1)\times p(t_2))$, and add $(t_1,t_2)$ to the list of non-monotonic leave-pairs.

We will apply this idea to a hedonic price model. 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 ([8]). The price of the good is determined by a combination of these valuations:

$$P = P(x_1, x_2, ..., x_n)$$

In the case study presented below we want to predict the house price given a number of characteristics. So the variables $x_1, x_2, ..., x_n$, correspond to the characteristics of the house. The data set consists of 116 observations of houses in the city of Den Bosch, which is a medium sized Dutch city with approximately 120,000 inhabitants. 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 1.
Of all 7021 distinct pairs of observations, 2217 are comparable, and 78 are non-monotonic. For the purpose of this study we have discretized the dependent variable (asking price) into the classes “below median” (fl. 347,500) and “above median”. In this way the regression problem is transformed into a binary classification problem. The regression problem is treated in the next section. After this discretization of the dependent variable only 9 pairs of observations are non-monotonic.

| 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 1: Definition of model variables

The tree algorithm used is in many respects similar to the CART program as described in ([2]). The program only makes binary splits and uses the Gini-index as splitting criterion.
Furthermore cost-complexity pruning is applied to generate a nested sequence of trees from which the best one is selected on the basis of test set performance. During tree construction, the algorithm records the minimum and maximum element for each node. These are used to check whether a tree is monotone.

In order to determine the effect of application of the monotonicity constraint we repeated the following experiment 100 times. The data set was randomly partitioned (within classes) into a training set of 60 observations and test set of 59 observations. The training set was used to construct a sequence of trees using cost-complexity pruning. From this sequence the best tree was selected on the basis of error rate on the test set (in case of a tie, the smallest tree was chosen). Finally, it was checked whether the tree was monotone and if not, the upper bound for the degree of monotonicity as described above was computed using algorithm 2.

Out of the 100 trees thus constructed, 61 turned out to be monotone and 39 not. The average misclassification rate of the monotonic trees was 14.9%, against 13.3% for the non-monotonic trees. Thus, the monotonic trees had a slightly worse classification performance.

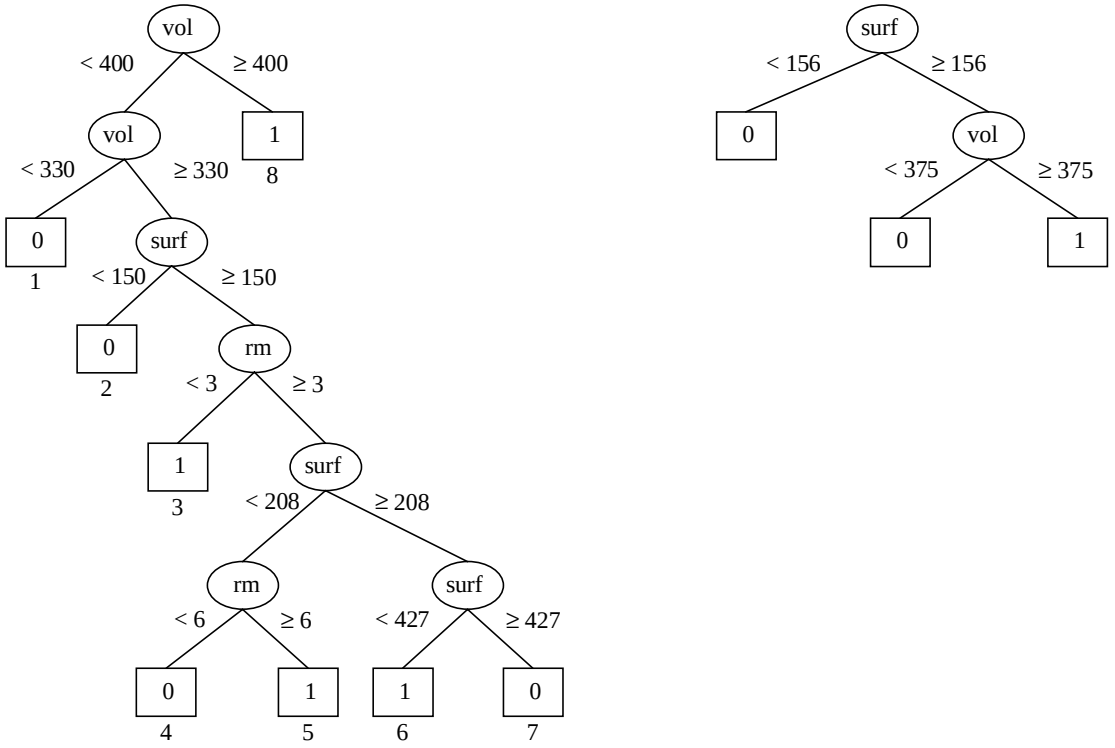


Figure 2 Example of a larger non-monotonic (left) and monotonic (right) tree.

Figure 2 depicts one of the 39 non-montonic trees (left part of the figure) and one of the 61 monotonic trees (right part of the figure) obtained in the experiment. The label 0 corresponds to prices below the median and label 1 to prices above median. It is easily verified that the leaf-pairs (3,4), (3,7), (5,7) and (6,7) of the left tree are non-monotone (the label of a leaf is given directly below it). The degree of non-monotonicity of this tree is only about 1%. The right tree is monotone and has only 3 leaf nodes. The error of the non-monotonic tree shown is 15.3%, and the estimated error of the monotonic tree 13.6%.

A two-sample t-test of the null hypothesis that monotonic and non-monotonic trees have the same classification error yielded a p-value of 0.0615 against a two-sided alternative. The average degree of non-monotonicity of the non-monotonic trees was about 1.7%, which is quite low, the more if we take into consideration that this is an upper bound. Another interesting comparison is between the average sizes of the trees. On average, the monotonic trees had about 3.13 leaf nodes, against 7.92 for the non-monotonic trees. Thus, the monotonic trees are considerably smaller and therefore easier to understand at the cost of only a slightly worse classification performance. The results are summarised in table 2.

| Trees | Monotonic | Non-monotonic |
|---|---|---|
| Total 100 | 61 | 39 |
| Average error rate | 14.9 % | 13.3 % |
| Average number of leaf nodes | 3.13 | 7.92 |
| Average degree of non-monotonicity | | 1.7 % |

Table 2: Comparison of monotonic and non monotonic trees.

**5. Monotonicity in regression and neural networks**.

The traditional method used in isotonic regression is the pool-adjacent violators algorithm ([16]). This method however only works in the one-dimensional case. A versatile non-parametric method that also works in higher dimensions is given in ([10]). The regression function we want to estimate is E($y \mid x$). We assume that E($y \mid x$) is monotonic in the sense of (3.1). Now if $t$ is any estimator of E($y \mid x$) we can make $t$ into a monotonic regression by simply defining:

$$G^+(x) = \max_{x' \le x} t(x'), \quad \text{and} \quad G^-(x) = \min_{x' \ge x} t(x') .$$

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

$$t_\alpha(x) = \sum_k y_k K\left(\frac{\| x - x_k \|}{\alpha}\right).$$

Noise in the data that may cause non-monotonic behaviour of the kernel smoother will be rectified by the maximisation and minimisation procedure in $G^+(x)$ and $G^-(x)$ respectively.
An undesirable side effect of the method is that this type of noise is accumulated in $G^+$ and $G^-$ .
One can avoid this effect by employing a parametric or semi-parametric estimator that is monotonic. In the sequel we will solve the regression problem by applying monotonic neural networks. The monotonicity costraint is built in the network by construction ([3]). It is well known that neural networks can be used to build flexible estimators. 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 ([15, chapter 4, section 3]). The class of monotonic neural networks applied here, have the advantage that the variance is decreased by the monotonicity constraint without increasing bias (if the problem is monotonic). 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

The method can be successfully combined with other regularisation methods. The implementation of monotonicity constraints in neural networks can be done in different ways. In [18] 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 multi-layer neural networks with non-negative weights. In can be shown that the elements of this class can approximate any monotonic increasing function, a sketch of the proof is given in [3]. For a neural network with one hidden layer and one output neuron we have

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

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 $\theta_i$ 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 developed is a modification of the standard backpropagation algorithm.
There are two canonical ways to enforce positive weights. The first one is to add a bias term to the error function of the neural network such that the negative weights are penalised :

$$E_m = E_s + \lambda * \sum_i (|w_i| - w_i)\,,$$

here $E_m$ is the modified error term, $E_s$ is the standard error and $\lambda$ is the scaling parameter. Note that only negative weights contribute to the extra penalty term. During the training the parameter $\lambda$ is gradually increased until all the weights are non-negative and the network is monotonic.
In the second method we set all negative weights to zero in each training step. This algorithm is described below.

**Algorithm 3: Training monotonic neural network.**

1. *Initialisation:* Start with a neural network configuration and set all weights and threshold (bias) levels to small, uniformly distributed random numbers.
2. *Presentation of the training patterns:* compute for each data pattern the output value using the current weight structure.
3. *Weights update:* using the standard backprop-algorithm determine the weight adjustments.
4. *Monotonicity correction:* change all negative weight values to zero.
5. *Repeat:* step 3 and 4 for all hidden layers.
6. *Repeat:* step 1 to 5 until the error is sufficiently small.

Note that only step number **4** differs from the standard backpropagation algorithm.

Both algorithms were implemented in MATLAB and have been extensively studied on artificially generated data sets.. Testing on artificially generated data sets is preferable to adjust and fine-tune the algorithms. The average performance of the algorithms do not differ, of course fluctuations occur in the error rate depending on the (random) choice of the starting vector. In this paper we do

not report on these studies [4] but prefer to illustrate the method in the case study described in section 4. Now the same problem of predicting the house price is treated as a regression problem with the same data set of 116 observations.

In the simulation study we compare ordinary neural networks and monotonic neural networks. Firstly an ordinary neural network is trained on the data set using 5 fold cross-validation. The error $R^2$ varies between 0.8089 and 0.9288. In the next step we trained ordinary neural networks 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 3.

| | 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 | 0.1 | 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 3: Performance of ordinary and monotonic 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.

**6. Conclusion.**

The goal of data mining is to derive valuable business knowledge from patterns in databases. In the majority of cases there is theoretical and domain dependent knowledge available. In this paper we have shown that the effectiveness of data mining systems can be substantially improved by using prior knowledge. We explicitly studied the framework of combining knowledge and data for different data mining algorithms, with a focus on decision 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 legislation or business policy rules. The methods are illustrated in a practical case study of predicting house prices.

**References.**


[1]: A. Ben-David, Monotonicity Maintenance in Information-Theoretic Machine Learning Algorithms, Machine Learning, 19**,** (1995) 29-43.

[2]: L. Breiman , J. H. Friedman, R.A. Olshen, C.T. Stone, Classification and Regression Trees, Wadsford, California, (1984).

[3]: H. A. M. Daniels, B. Kamp, Application of MLP networks to bond rating and house pricing, Neural Computation and Applications, 8, (1999) 226-234.

[4]: H.A.M. Daniels, B. Kamp, W.J. Verkooijen, Application of neural networks to house pricing and bond rating, Internal report, CentER DP series on Management Science, n0 9796, (1997) 1-18.

[5]: U. Fayyad, G. Piatetsky- Shapiro, P. Smyth, R. Uthurusamy (eds.), Advances in knowledge discovery and data mining, AAAI Press, (1996).

[6]: A. Feelders, H.A.M. Daniels, M. Holsheimer, Methodological and practical aspects of data mining, Information & Management, 37, (2000) 271-281.

[7]: J. Han, M. Kamber, Data-Mining: Concepts and Techniques, Morgan Kaufmann Publishers, (2001).

[8]: O. Harrison, D. Rubinfeld, Hedonic prices and the demand for clean air, Journal of Environmental Economics and Management, 53, (1978) 81-102.

[9]: E. Leamer, Specification Searches, ad hoc inference with non-experimental data, New York, Wiley, (1978).

[10]: H. Mukarjee, S. Stern, Feasible non-parametric estimation of multi-argument monotone functions, Journal of the American Statistical Association, 89, no.425, (1994) 77-80.

[11]: M. Nunez, The use of background knowledge in decision tree induction, Machine Learning, 6, (1991) 231-250.

[12]: S.C. Park, S. Piramuthu, M.J. Shaw, Dynamic rule refinement in knowledge- based data mining systems, Decision Support Systems 31 (2) (2001) 205-222.

[13]: R. Potharst, Classification using decision trees and neural nets, Erasmus Universiteit Rotterdam, SIKS Dissertation Series No. 99-2, (1999).

[14]: J.R. Quinlan, C4.5 Programs for machine learning , Morgan Kaufmann, California, (1993).

[15]: B.D. Ripley, Pattern recognition and neural networks, Cambridge University Press, (1996).

[16]: T. Robertson, F. Wright, R.L.Dykstra, Order restricted statistical inference, New York, Wiley, (1988).

[17]: M.J. Shaw, C. Subramaniam, G.W. Tan, M.E. Welge, Knowledge management and data mining for marketing, Decision Support Systems 31 (2001) 127-137.

[18]: S.Wang, A neural network method of density estimation for univariate unimodal data, Neural Computation & Applications, 2, (1994) 160- 167.