Combining domain knowledge and data for house price modelling with classification trees and neural networks

Hennie Daniels[1] and Ad Feelders[2]

[1] Tilburg University, CentER for Economic Research
Tilburg, The Netherlands
and
Erasmus University Rotterdam, ERIM
Rotterdam, The Netherlands
H.A.M.Daniels@kub.nl
[2] Utrecht University, Department of Computer Science
Utrecht, The Netherlands
ad@cs.uu.nl

Abstract. The combination of data and domain knowledge is a topic of ongoing interest in data mining and data analysis in general. In this paper we discuss methods for implementing monotonicity constraints; a very common form of prior knowledge in economic decision problems. We show how classification tree algorithms and neural network algorithms can be adapted to allow for the inclusion of such prior knowledge. The methods are illustrated on a model for house pricing.

1 Introduction

The combination of data and domain knowledge is a topic of ongoing interest in model construction in many application areas. In theory there are two extreme situations that may occur with respect to the availability of domain knowledge in data analysis. The first is that no prior knowledge whatsoever is available, the second is that the relationship to be modelled is known with certainty, up to a limited number of parameters. Both extremes are unlikely to occur in practice.

In econometrics one usually assumes that a *qualitative* specification of the relationship to be modelled is known in advance. 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 however does not exactly conform to this textbook approach but is often characterised by so called *specification searches* [Lea78]. If extensive specification searches are performed, standard econometric model testing is no longer valid however. In the data mining community this problem is well-known, 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 initial collection and furthermore flexible functions are used.

In this paper we position ourselves somewhere between these extremes. We focus on the implementation of a specific kind of domain knowledge that is typical in economic decision problems, namely knowledge concerning 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 we would expect a negative relationship between price and demand. The strength of this relationship and the precise functional form are hardly ever indicated by economic theory however.

This paper is organised as follows. In section 2 we define the monotonicity constraint in both the regression and classification context. In section 3 we discuss the incorporation of the monotonicity constraint in classification tree algorithms, and describe its application to the prediction of house prices. The implementation of monotonicity in neural networks as well as its application to house price prediction is discussed in section 4. Finally, we draw a number of conclusions.

2 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. A well-known example is

the dependence of labour wages as a function of age and education [MS94]. In loan acceptance the decision rule should be monotone with respect to income for example, i.e. it would not be acceptable that an applicant with high income is rejected, whereas another applicant with low income and otherwise equal characteristics is accepted. 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 [HR78]. 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 $\mathcal{Y}$ and the vector of independent variables is $\mathbf{x}$ and takes values in $\mathcal{X}$. In the applications discussed here, $\mathcal{Y}$ is a one-dimensional space of prices or classes and $\mathcal{X}$ is a n-dimensional space of characteristics of products or customers for example.

Furthermore we assume that we have a dataset $(y^p, \mathbf{x}^p)$ of points in $\mathcal{Y} \times \mathcal{X}$, which can be considered as a random sample of the joint distribution of $(y, \mathbf{x})$. In a regression problem we want to estimate $E(y|\mathbf{x})$. The monotonicity constraint for $E(y|\mathbf{x})$ can be formulated as:

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

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

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

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

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

In the next section we discuss the implementation of monotonicity in 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 4 we implement monotonicity in regression and neural network models. The same house-pricing model is used for testing.

3 Monotonic decision trees

Tree-based algorithms such as CART [BFOS84] and C4.5 [Qui93] are very popular in data mining. 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 (2). 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 assigned 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. Figure 1 shows an example of a pair of non-monotonic leaf nodes. A dataset $(y^p, \mathbf{x}^p)$, is called monotone if

$$\mathbf{x}^1 \geq \mathbf{x}^2 \Rightarrow y^1 \geq y^2,$$

for all possible combinations of data points $\mathbf{x}^1, \mathbf{x}^2$.

Potharst [Pot99] 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 [BD95], who proposes a splitting rule that includes a non-monotonicity index in addition to the usual impurity measure. This non- monotonicity index 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 $p(t_1) \times p(t_2)$, where $p(t_i)$ denotes the proportion of cases in leaf 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

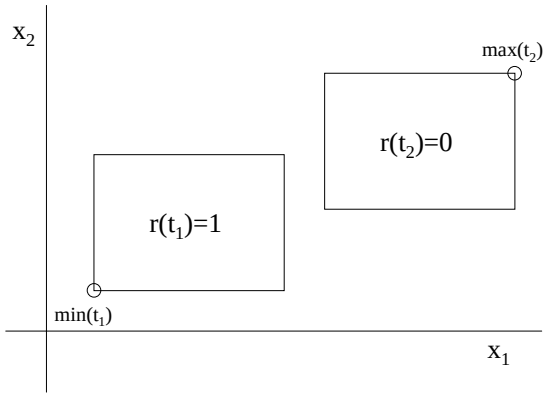

Fig. 1. An example of a pair of non-monotonic nodes

reader should note that $p(t_1) \times p(t_2)$ 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 use of a non-monotonicity index in determining 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. This approach has the additional advantage that one can estimate to what extent the assumption of monotonicity is correct.

We 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 [HR78]. The price of the good is determined by a combination of these valuations:

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

In the case study presented below we want to predict the house price given a number of characteristics. So the variables , 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.

The tree algorithm used is in many respects similar to the CART program as described in [Bre, 84]. 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

Table 1. Definition of model variables

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

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. The results are summarised in table 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. 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.

Table 2. Comparison of monotonic and non-monotonic trees

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%

4 Monotonicity in regression and neural networks

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]. The regression function we want to estimate is $E(y|\mathbf{x})$. We assume that $E(y|\mathbf{x})$ is monotonic in the sense of (1).

Now if t is any estimator of $E(y|\mathbf{x})$ 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 $E(y|\mathbf{x})$. 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^-.

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 [DK99]. 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 (see [Rip96] 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 relation 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 (see e.g. [DK99]). 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 [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 multi-layer neural networks with non-negative weights. It can be shown that the elements of this class can approximate any monotonic increasing function, a sketch of the proof is given in [DK99].

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 data sets. The performance differs slightly but not significantly. Testing on artificially generated data sets is preferable to adjust and fine-tune the algorithms, since everything is under control. In this paper we do not report on these studies but prefer to illustrate the method in the case study described in section 3. 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. As a check for monotonicity we computed the monotonicity indices for each of the variables. This is in line with economic intuition. 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.

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.

5 Conclusion

The goal of data mining is to derive valuable knowledge from databases. In many cases there is theoretical and domain dependent prior knowledge available. We studied ways to incorporate knowledge concerning monotonicity of relations in tree-based algorithms and neural net-

Table 3. Performance of normal and monotonic networks

	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

works. Application to the prediction of house prices resulted in some encouraging results.

With the classification trees we obtained much simpler models at the expense of only a slight decrease in classification performance. This is especially of interest in applications for which monotonicity of the resulting model is an absolute requirement, as might be the case for acceptance/rejection decisions. A nice feature of our resampling based approach is that it also provides an indication of the validity of the monotonicity constraint. If all the resulting trees are highly non-monotonic, we have to question our prior "knowledge".

For the neural networks the monotonicity constraint resulted in a reduced tendency to overfit the data, and a better out-of-sample performance.

References

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

[BFOS84] L. Breiman, J.H. Friedman, R.A. Olshen, and C.T. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, California, 1984.

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

[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.

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

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

[Qui93] J.R. Quinlan. *C4.5 Programs for Machine Learning.* Morgan Kaufmann, San Mateo, California, 1993.

[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.