Monotone Oblique Decision Trees

Maarten Al Sadawi and Ad Feelders

Department of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands
maailani60@gmail.com,a.j.feelders@uu.nl

Abstract. In many applications of supervised learning the response variable is known a priori to be increasing (or decreasing) in one or more of the features. Such relations between response and feature are called monotone. Monotonicity may also be a desirable property of a decision model for reasons of explanation, justification and fairness. Because the monotonicity constraint is quite common in practice, many data analysis techniques have been adapted to be able to handle such constraints. For example, several algorithms have been developed for learning monotone decision trees. These algorithms, however, only consider splits on single features, giving rise to axis-parallel splits, and a partitioning of the feature space into rectangular areas. Oblique decision trees also allow linear combination splits, and may therefore produce more compact models, but no algorithms exist to enforce monotonicity for this more complex partitioning of the feature space. We present such an algorithm and evaluate its performance through experiments on artificial as well as real life data.

Keywords: classification trees · regression trees · monotonicity constraints · oblique decision trees

1 Introduction

In many applications of supervised learning the response variable is known a priori to be increasing (or decreasing) in one or more of the features. For example, all else equal, companies with a higher debt ratio are more likely to go bankrupt (increasing relationship), and used cars with a higher mileage are less expensive (decreasing relationship). Such relations between response and feature are called monotone. Monotonicity may also be a desirable property of a decision model for reasons of explanation [10], justification and fairness. For example, it would be considered unfair if applicant A scores better than B on all criteria, but B receives the more favorable decision. Since in monotone models the relation between a feature and the response is either increasing or decreasing, regardless of the values of the other features, it is easier to isolate the features that contributed to, or counteracted, a change in the response. This makes it easier to provide (contrastive) explanations of model decisions. Moreover, monotonicity may improve model acceptance by domain expert. For example, Pazzani et al.[11], show that rules learned with monotonicity constraints were significantly more acceptable to medical experts than rules learned without the monotonicity restrictions.

Because the monotonicity constraint is quite common in practice, many data analysis techniques have been adapted to be able to handle such constraints. For example, several algorithms have been developed for learning (partially) monotone decision trees, see e.g. [17,2,1,12]. These algorithms, however, only consider splits on single features, giving rise to axis-parallel splits, and a partitioning of the feature space into rectangular areas. Oblique decision trees also allow linear combination splits, and may thus produce more compact models. Pei et al. [13] present an algorithm for monotone oblique decision trees, but their algorithm does not guarantee a globally monotone model. Only local monotonicity (monotone splits) is enforced, but as the authors themselves note, a tree with locally monotone splits can still be globally non-monotone. This is illustrated in figure 1. In this figure, the splits are numbered, where split ① is the split in the root node. All points below that line are initially assigned to class 0, and all points above the line are initially assigned to class 1. This split does not create any monotonicity violations (smaller points getting a higher label than larger points). Split ② further splits up the area below the line, and split ③ further slits up the area above the line. Splits ② and ③ are locally monotone as well, but the combination of splits produces a globally non-monotone prediction rule because ultimately, some smaller points get assigned a higher class label than some larger points. It should be noted that this behavior can also occur in trees with exclusively axis-parallel splits. To make the tree globally monotone, one could for example relabel the smaller area labeled 1 to 0, which would effectively prune away the second split.

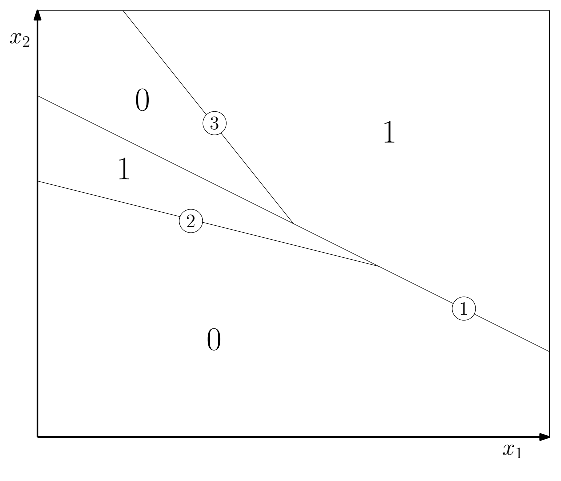


Fig. 1: Locally monotone splits that produce a globally non-monotone tree in a binary classification problem. The splits are numbered, where the split in the root node has number ①. See the text for further explanation.

The novel contribution of this paper is that we present the first algorithm that is able to enforce global monotonicity for oblique decision trees.

This paper is organized as follows. In section 2 we introduce a number of basic concepts and notation. Section 3 describes the proposed algorithm for learning monotone oblique decision trees. We evaluate the predictive performance of the algorithm on artificial and real data sets in section 4. Finally, section 5 concludes.

2 Preliminaries

Let $\mathcal{X}$ be a feature space $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_p$ consisting of vectors $\mathbf{x} = (x_1, x_2, \ldots, x_p)$ of values on p features. We assume that each feature takes values x_i in a linearly ordered set $\mathcal{X}_i$. Assuming an increasing relation between each feature and the response, the partial ordering $\preceq$ on $\mathcal{X}$ is defined to be the ordering induced by the order relations of its coordinates $\mathcal{X}_i$:

$$\mathbf{x} = (x_1, x_2, \ldots, x_p) \preceq \mathbf{x}' = (x_1', x_2', \ldots, x_p') \Leftrightarrow \forall i : x_i \leq x_i'.$$

Note that if there is instead a decreasing relationship between a feature and the response, we can simply invert the feature values to obtain an increasing relationship.

A monotone prediction rule is a function $f : \mathcal{X} \to \mathbb{R}$ for which

$$\mathbf{x} \preceq \mathbf{x}' \Rightarrow f(\mathbf{x}) \leq f(\mathbf{x}') \tag{1}$$

for all instances $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$.

A data set $\{\mathbf{x}_i, y_i\}_{i=1}^n$ is monotone if for all i, j we have $\mathbf{x}_i \preceq \mathbf{x}_j \Rightarrow y_i \leq y_j$. We say two data points $\mathbf{x}_i$ and $\mathbf{x}_j$ are comparable if one dominates the other, that is, if $\mathbf{x}_i \preceq \mathbf{x}_j$ or $\mathbf{x}_j \preceq \mathbf{x}_i$. Otherwise they are incomparable. Note that for a pair of incomparable points, the constraint in equation (1) is vacuously satisfied, since the antecedent is false.

Next, we define the isotonic regression [15], which is used in the sequel to make trees globally monotone. Let $Z = \{z_1, z_2, \ldots, z_m\}$ be a set of constants and let $\preceq$ be a partial order on Z. Any real-valued function f on Z is isotonic with respect to $\preceq$ if, for any $z, z' \in Z, z \preceq z'$ implies $f(z) \leq f(z')$. We assume that each element z_i of Z is associated with a real number $g(z_i)$; these real numbers typically are estimates of the function values of an unknown isotonic function on Z. Furthermore, each element of Z has associated a positive weight $w(z_i)$ that typically indicates the precision of this estimate. An isotonic function g^* on Z now is an isotonic regression of g with respect to the weights w and partial order $\preceq$ if and only if it minimizes the sum

$$\sum_{i=1}^{m} w(z_i)[f(z_i) - g(z_i)]^2 \tag{2}$$

in the class of isotonic functions f on Z. In words, $g^*(z)$ makes the smallest possible adjustments to the given values $g(z)$ (in a weighted least squares sense) so as to make the result monotone with respect to the given partial order. It has been shown that g^* exists and is unique [15].

3 Monotone Oblique Decision Trees

In this section we describe the main components of the proposed algorithm for learning monotone oblique decision trees. The current implementation is restricted to binary classification and regression. Ordinal classification with more than two classes is left for future research. For purposes of experimental comparison, we consider both local and global monotonicity constraints. The local constraint was proposed by Pei et al. in [13]. Implementation of the global constraint is our new contribution. Since each constraint (local or global) can be either enforced or not, the algorithm can run in 4 different constraint configurations.

3.1 Tree Growing

The splits considered in each node are as follows:

1. If there is no local monotonicity constraint, compute $\hat{y} = w_0 + \mathbf{w}^\top \mathbf{x}$ using ordinary least squares (OLS), and create the linear combination variable $\mathbf{w}^\top \mathbf{x}$.
2. If there is a local monotonicity constraint, compute $\hat{y} = w_0 + \mathbf{w}^\top \mathbf{x}$ using non-negative least squares, and create the linear combination variable $\mathbf{w}^\top \mathbf{x}$.
3. For each feature x (including the derived linear combination variable), consider the splits $(x \leq b, x > b)$, where b is any value halfway in-between two consecutive data values of x in their sorted order.

It should be noted that in the case of binary classification ($y \in \{0, 1\}$), the OLS estimates $\mathbf{w}$ are proportional to the weights of the linear discriminant function [8]. Since the intercept w_0 is ignored altogether, this means we obtain the same optimal split on the derived variable for OLS and linear discriminant analysis (LDA). Also note that the only effect of the local constraint is that non-negative least squares regression is used instead of OLS to construct the linear combination split. This means that all coefficients in the linear combination are restricted to be positive. Typically, this leads to less complex splits, since features that would get a negative coefficient using OLS tend to drop out (get a zero coefficient) using non-negative least squares.

For binary classification we use the gini-index impurity function. For regression problems we use the sum of squared deviations from the mean as impurity measure. We perform the split that maximizes impurity reduction.

The minimum number of observations required in a node for it to be allowed to be split is set to 2, so tree growing tends to produce a (highly) overfitted tree, denoted by $T_{\max}$.

3.2 Enforcing Global Monotonicity

Consider a tree T that has been grown on the training data. The usual rule is to predict the mean response in the leaf nodes of the tree. This prediction rule may however not be monotone, that is, it may fail to satisfy equation (1). Our approach is to make it monotone by adjusting the leaf predictions as little as possible (in the weighted least squares sense of equation (2)). To do so, we first have to establish all violations of monotonicity between predictions made in pairs of leaf nodes. A pair of leaf nodes (t, t') violates equation (1) if

1. There are $\mathbf{x} \in t$, $\mathbf{x}' \in t'$ such that $\mathbf{x} \preceq \mathbf{x}'$, and
2. $\bar{y}(t) > \bar{y}(t')$,

where $\bar{y}(t)$ denotes the mean y value of all cases that fall into node t. In non-oblique decision trees, the first condition is easily verified by recording the minimal and maximal elements of each node (rectangle), and testing whether $\min(t) \prec \max(t')$ [17].

This is no longer possible in oblique trees. In an oblique tree, a leaf node corresponds to a set of linear inequalities of the form: $w_1x_1+w_2x_2+\ldots+w_px_p \leq b$ or , $w_1x_1+w_2x_2+\ldots+w_px_p > b$. Note this corresponds to an axis-parallel split if all weights but one are equal to zero. Leaf nodes contain the points $\mathbf{x}$ that satisfy all inequalities on the path from the root node to the leaf. Hence, to verify the first condition, we have to determine whether or not a collection of linear inequalities has a solution. The constraint satisfaction system we use for this purpose is an API, written in Python, which is based on a high performance theorem prover called Z3 [6].

Let $\tilde{T}$ denote the collection of leaf nodes of tree T, $\bar{y}(t)$ the average y value of all observations that fall into node t (the unconstrained prediction in node t), and $n(t)$ the number of observations that fall into node t. Let us write $t \precsim t'$ if there exist $\mathbf{x} \in t$, and $\mathbf{x}' \in t'$ such that $\mathbf{x} \preceq \mathbf{x}'$. If there is a monotonicity violation between any pair of leaf nodes, that is, $t \precsim t'$ and $\bar{y}(t) > \bar{y}(t')$, monotonicity is restored by adjusting the leaf predictions as little as possible. This is achieved by performing the isotonic regression on $(Z = \tilde{T}, \precsim)$ with values $g(t) = \bar{y}(t)$ and weights $w(t) = n(t)$. We use the generic notation $\hat{y}(t)$ for the prediction made in node t. Note that the number of observations falling into a leaf node is used as a weight in the weighted sum of squares error function in equation (2). For example, if two leaf nodes, t and t', violate monotonicity, and setting

$$\hat{y}(t) = \hat{y}(t') = \frac{n(t)\bar{y}(t) + n(t')\bar{y}(t')}{n(t) + n(t')}$$

would make the tree monotone, then this would be the solution produced by the isotonic regression: their weighted average is assigned to both leaf nodes. In general, the isotonic regression will partition the set of leaf nodes into a number of blocks on which the solution is constant and equal to the weighted average of the predictions of the leaves contained in that block.

Note that the relation $\precsim$ on the leaf nodes is actually not a partial order, since it is neither transitive, nor antisymmetric. We can however afford this slight abuse of notation, since the isotonic regression will automatically satisfy the transitive closure of $\precsim$, and hence we can refrain from stating these constraints explicitly. The complexity of the isotonic regression algorithm we used [16] is $O(|\tilde{T}|^3)$.

3.3 Pruning

To avoid overfitting, we apply cost-complexity pruning [4]. This yields the pruning sequence:

$$T_1 < T_2 < \cdots < \{t_1\},$$

where each tree in the sequence is a pruned subtree of the previous one, T_1 is the smallest pruned subtree of $T_{\max}$ with the same in-sample prediction error, and t_1 is the root node. The best tree from this sequence is selected through cross-validation. If global monotonicity is required, the following procedure is applied to the selected tree:

(1) If there is a monotonicity violation, make the tree monotone by performing the isotonic regression as described in section 3.2. Otherwise stop.
(2) In case of a binary classification problem:
  (a) Every pair of leaf nodes (t_ℓ, t_r) with common parent t that predict the same class label (i.e. $\hat{y}(t_\ell)$ and $\hat{y}(t_r)$ are on the same side of 0.5) is pruned back to its parent. The parent t gets assigned prediction $\hat{y}(t) = \bar{y}(t)$.
  (b) Return to step (1).

There is a choice of when to enforce the global constraint in the model selection process. Here, we have chosen to first select the tree from the pruning sequence with lowest cross-validation error, and only then make this tree monotone. Another option is to first make every tree in the pruning sequence monotone, and then select the tree with lowest cross-validation from the resulting sequence. The latter option is obviously computationally more expensive.

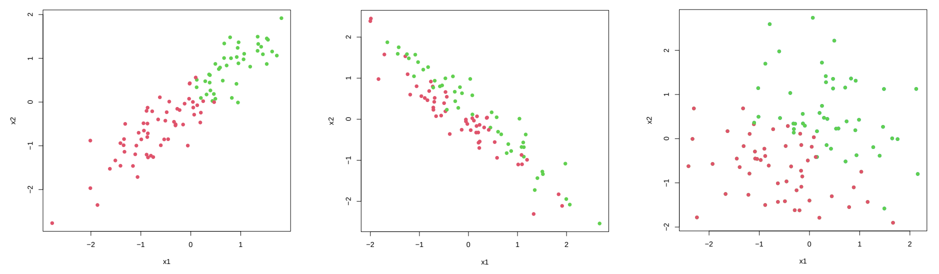
4 Experimental Evaluation

In this section we present the results of the experiments we performed on artificial binary classification data (section 4.1), real regression data (section 4.2), and real binary classification data (section 4.3). To study the separate effects of enforcing the local and global monotonicity constraint, we run the tree algorithm in four different configurations. It should be noted that enforcing the local monotonicity constraint only means that the weights in the linear combination splits are determined by non-negative least squares instead of ordinary least squares. This avoids linear combination splits containing weights with opposite signs, that would force one to predict the same value in both child nodes to prevent a monotonicity violation. In addition, we would like to point out that we only consider relatively small training sets in the experiments, because for large data sets the monotonicity constraint (if correct) will almost always be satisfied already by the unconstrained model. Hence, we only expect a better predictive performance for relatively small training samples.

4.1 Artificial Data

To study the relationship between properties of the data and algorithm performance, we generate artificial data for the binary classification problem. The artificial data sets are constructed in two phases. First, a data set without labels is generated, and then random monotone labelings per data set are produced such that the label distribution is sufficiently balanced.

Three types of data sets are drawn from a multivariate normal distribution, one with high positive (0.9) correlation between the features (from now on referred to as the *positive dataset*), one with zero correlation between the features (the *zero dataset*) and one with high negative correlation (-0.9) between the features (the *negative dataset*). For the latter, we create two groups of features of the same size, such that the features are positively correlated (0.9) within their group, but negatively correlated (-0.9) between the groups. The number of features is either 2, 6 or 10. We generate a total of 2000 data points for each combination of correlation value and number of features.



(a) Positive correlation. (b) Negative correlation. (c) Zero correlation.

Fig. 2: Illustrative datasets of 100 data points and 2 features each, with different degrees of correlation. The green dots are observations from class 1. The class labels were randomly generated using the Propp-Wilson algorithm

For each set of data points, multiple random monotone labelings are generated using the Propp-Wilson algorithm with sandwiching [14]. Only the sufficiently balanced labelings are used for the experiments: the relative frequency of the majority class is allowed to be at most 60%. Note that the labeling has no structure at all, apart from being monotone and sufficiently balanced. Each labeled dataset is then split into a training set (of size 50, 100, or 150) and a test set (all remaining data points). Figure 2 shows three example datasets with different degrees of correlation, and monotone labelings generated with the Propp-Wilson algorithm.

Next, we introduce non-monotone noise to the training sets and test sets. The amount of noise is measured by the number of data points that need to be relabeled to restore monotonicity, and is introduced as follows. First we find a collection of comparable pairs with the same label. Consider such a pair $(\mathbf{x}, \mathbf{x}')$

with $\mathbf{x} \preceq \mathbf{x}'$. If both labels are 0, we change the label of $\mathbf{x}$ to 1. Likewise, if both labels are 1, we change the label of $\mathbf{x}'$ to 0. By choosing the flipped pairs such that they are mutually disjoint, it is guaranteed that the number of labels that need to be flipped to restore monotonicity is precisely equal to the number of labels that were flipped.

We flip the labels of 5% of the training set, respectively test set. We use relatively small training sets of size 50, 100, and 150, because the monotonicity constraint is likely to be more effective in those cases. For larger training sets, the monotonicity constraint is more likely to be satisfied by the data based (unconstrained) estimates already, in which case imposing the constraint won't make any difference.

Table 1 contains the accuracy on the test set of the different configurations of the algorithm (LG, $\bar{L}G$, $L\bar{G}$ and $\bar{L}\bar{G}$), different correlations (Positive, Zero, Negative), number of features (2,6, and 10) and training set sizes (50, 100 and 150). Here L indicates that the local monotonicity constraint is enforced, and G indicates that the global constraint is enforced.

From table 1 we conclude that, all else equal, accuracy tends to increase with correlation between the features. For example, for LG with 2 features and $N = 150$, the accuracies are $(0.84, 0.90, 0.92)$, for negative, zero, and positive correlation respectively. This is explained by the fact that the monotonicity constraint on the labels is more restrictive with positively correlated features than with negatively correlated features, leading to a stronger correlation between features and class label. This stronger correlation obviously leads to better predictive performance. This phenomenon is also suggested by the example data sets shown in figure 2. To validate the observation that accuracy tends to increase with correlation between the features, we conduct a statistical test. For each fixed degree of correlation, there are 3 values for the number of features, 4 constraint configurations and 3 different training set sizes, constituting a total of 36 accuracy values. We set $\alpha = 0.05$. The p-value of the Friedman test [7] is 1.04×10^{-14}. The p-values of the post-hoc Nemenyi test establish that the performance on positive datasets is significantly better than on zero or negative datasets, and the performance on zero datasets is significantly better than on negative datasets ($p = 0.001$).

Likewise, we note from table 1 that, all else equal, accuracy tends to decline with the number of features. To give a pronounced example, if we look at LG with $N = 50$, and no correlation between the features, the accuracies are $(0.88, 0.64, 0.51)$ for 2, 6, and 10 features respectively. This trend can be explained by the same phenomenon, as more features means fewer comparable pairs, with the consequence that the monotone labeling requirement imposes less structure on the data. For each fixed number of features there are 3 degrees of correlation, 4 constraint configurations and 3 different training set sizes, constituting a total of 36 accuracy values. The p-value of Friedman test is 2.32×10^{-16}. The p-values of the post-hoc Nemenyi test establish that the performance on 2 features is significantly better than on 6 or 10, and the performance on 6 features is significantly better than on 10 ($p = 0.001$).

Table 1: Test set accuracy for artificial datasets.

	nfeat	ntrain	LG	$\bar{L}G$	$L\bar{G}$	$\bar{L}\bar{G}$
Positive	2	50	0.8883	0.8831	0.8875	0.8822
		100	0.9	0.9	0.9	0.9
		150	0.921	0.917	0.921	0.917
Positive	6	50	0.86	0.74	0.86	0.84
		100	0.873	0.678	0.872	0.858
		150	0.87	0.61	0.87	0.86
Positive	10	50	0.8385	0.7017	0.8398	0.8125
		100	0.85	0.64	0.85	0.83
		150	0.86	0.58	0.86	0.83
Zero	2	50	0.8786	0.8776	0.8786	0.8781
		100	0.892	0.892	0.891	0.891
		150	0.9	0.9	0.9	0.9
Zero	6	50	0.643	0.56	0.641	0.629
		100	0.655	0.63	0.655	0.657
		150	0.6682	0.6495	0.6678	0.665
Zero	10	50	0.508	0.498	0.507	0.503
		100	0.5174	0.5058	0.5134	0.5129
		150	0.515	0.509	0.518	0.512
Negative	2	50	0.813	0.828	0.826	0.83
		100	0.8455	0.8445	0.8444	0.8435
		150	0.8417	0.8411	0.8434	0.8427
Negative	6	50	0.581	0.532	0.583	0.572
		100	0.583	0.55	0.59	0.587
		150	0.587	0.555	0.584	0.577
Negative	10	50	0.497	0.502	0.494	0.496
		100	0.4945	0.5086	0.4971	0.4999
		150	0.506	0.511	0.51	0.507

Table 2: The p-values of the post-hoc Nemenyi test for different constraint configurations with $n = 50$ (left) and $n = 150$ (right).

	LG	$\bar{L}G$	$L\bar{G}$
$\bar{L}G$	0.052		
$L\bar{G}$	0.9	0.13	
$\bar{L}\bar{G}$	0.26	0.88	0.46

	LG	$\bar{L}G$	$L\bar{G}$
$\bar{L}G$	0.082		
$L\bar{G}$	0.9	0.018	
$\bar{L}\bar{G}$	0.46	0.77	0.18

Comparing the performance of the four different constraint configurations for training set size 50, we find the p-value of the Friedman test is 0.03. The post-hoc Nemenyi test however could not find significant difference(s), see table 2 (left part), but we point out that the p-value of the test between LG and $\bar{L}G$ is the smallest of all. Inspection of table 1 tells us that LG consistently has higher accuracy than $\bar{L}G$, except when correlation is negative.

Comparing the performance of the four different constraint configurations for training set size 100, we find the p-value of the Friedman test is 0.31, so we could not reject the null hypothesis that all four constraint combinations perform equally well.

Finally, for training set size 150, the p-value of the Friedman test is 0.0053. See table 2 (right part) for the p-values of the post-hoc Nemenyi test. Here, only enforcing the local constraint ($L\bar{G}$) is significantly better than only enforcing the global constraint ($\bar{L}G$).

4.2 Real Regression Data

Table 3 lists the regression datasets used in the experiments, together with their number of observations and features after pre-processing.[1] We selected features for which the monotonicity constraint was plausible based on common sense, or consensus within the application domain. The expected signs of the monotonicity constraints were confirmed by inspecting the signs of the coefficients in a linear regression fitted to the data [9]. To illustrate, we consider the Wages dataset. The target attribute of this dataset is the hourly wage in dollars of a person. The selected features are "educ" (years of education), "exper" (years of potential labor force experience), "tenure" (years with the current employer), "female" (1 if female, 0 otherwise), "nonwhite" (0 if white, 1 otherwise), "numdep" (number of dependents), and "married" (1 if married, 0 otherwise). We expected, "female" and "nonwhite" to have a decreasing monotone relation with the target, which was confirmed by their negative coefficients in the linear regression model. They have therefore been inverted to "male" and "white". This way, all selected features are expected to have an increasing monotone relation with the target.

Table 4 shows the MSE for all constraint combinations, for training set sizes 50, 100, and 150 respectively.

We note that in general the combined application of the local and global constraint performs best: it has the lowest MSE in 4 out of the 6 datasets (for train set size 100 even 5 out of 6). The results confirm the utility of the local constraint, as the combinations that do not apply the local constraint never win

[1] The pre-processed datasets and Python code can be found on `https://github.com/ZazeyManda/IDT-oblique.git`

[2] `https://www.kaggle.com/datasets/akshaydattatraykhare/data-for-admission-in-the-university`

[3] `https://www.kaggle.com/datasets/uciml/autompg-dataset`

[4] `http://qed.econ.queensu.ca/jae/2006-v21.3/stengos-zacharias/`

[5] `https://jse.amstat.org/jse_data_archive.htm`

[6] `https://rdrr.io/cran/wooldridge/man/wage1.html`

[7] `https://www.kaggle.com/datasets/photosho/house-prices-for-the-city-of-windsor-canada`

Table 3: Regression datasets after preprocessing.

Dataset	N	nfeat
Admission[2]	400	5
AutoMPG[3]	392	7
Computer[4]	6259	9
Kuiper[5]	804	5
Wages[6]	526	7
Windsor[7]	546	11

Table 4: Test set MSE for regression datasets.

	n	LG	$\bar{L}G$	$L\bar{G}$	$\bar{L}\bar{G}$
Adm.	50	0.0072	0.0114	0.0093	0.0109
	100	0.0055	0.0072	0.0073	0.0070
	150	0.0056	0.0072	0.0062	0.0066
Auto	50	17.53	26.08	17.53	18.38
	100	11.29	25.38	11.29	12.02
	150	11.99	60.65	10.55	12.2
Comp.	50	142200	351550	212403	218503
	100	94112	106919	97988	116586
	150	87466	92163	88271	95291
Kuip.	50	101475000	103377100	64724450	69963970
	100	106261200	85241970	49858290	50385930
	150	103304700	103198100	40536170	38513820
Wage	50	10.86	34.46	10.74	13.03
	100	9.14	14.33	9.7	13.3
	150	9.11	16.83	9.41	10.61
House	50	363860200	420322200	412652000	574990300
	100	311893800	415459900	325382200	413842200
	150	284492000	334268700	299826600	335350300

(the only exception being Kuiper with $n = 150$). The second best option is to apply the local, but not the global constraint, which wins (or ties for a win) 6 times in total. All in all, we conclude that application of the monotonicity constraints is beneficial.

Comparing the 4 constraint configurations for a training set size of 50, the p-value of the Friedman test equal to 0.0035. As this is smaller than $\alpha = 0.05$, it implies that there is a statistically significant difference between some MSE values. See table 5 (left) for the results of the post-hoc Nemenyi test. We can conclude that there is significant difference between configurations $\bar{L}G$ and LG. If we look at the individual MSE values for both configurations in table 4, we can conclude that the MSE values are smaller for LG and thus it performs significantly better than the $\bar{L}G$. Similarly, configuration $\bar{L}G$ is significantly worse based on MSE than $L\bar{G}$. From this, we infer that the local monotonicity constraint is more critical than the global constraint. Even though we found no significant difference, we still see that the MSE of configuration LG is lower than the MSE of $\bar{L}G$.

For a training set size of 100, the p-value of the Friedman test is equal to 0.051. This is larger than our α and thus we cannot reject H_0 in this case.

For a training set size of 150 the p-value of the Friedman test is 0.035. We proceed with the post-hoc Nemenyi test to determine the significant differences, see table 5 (right). Unfortunately, the post-hoc Nemenyi test could not find any significant differences (α still at 0.05) between the MSE values. This is because the test is known to be less accurate than the Friedman test [5]. We reject H_0 but cannot give a conclusive answer which constraint configuration outperforms another, although one can read from table 4 that LG has lower MSE values than $\bar{L}G$. So the local monotonicity constraint results in lower MSE on the test sample.

Table 5: The p-values from the post-hoc Nemenyi test for regression with $n = 50$ (left), and $n = 150$ (right).

	LG	$\bar{L}G$	$L\bar{G}$
$\bar{L}G$	0.014		
$L\bar{G}$	0.9	0.014	
$\bar{L}\bar{G}$	0.23	0.66	0.23

	LG	$\bar{L}G$	$L\bar{G}$
$\bar{L}G$	0.07		
$L\bar{G}$	0.9	0.11	
$\bar{L}\bar{G}$	0.28	0.9	0.4

4.3 Real Binary Classification Data

Table 6 displays some basic information for the binary classification datasets, after pre-processing, that we used in the experiments.[8] We illustrate the pre-processing performed by shortly discussing the Bankruptcy dataset. The class

[8] The pre-processed datasets and Python code can be found on `https://github.com/ZazeyManda/IDT-oblique.git`.

label indicates whether a company went bankrupt (1) or not(0). The following financial ratios were selected as features: 'Debt ratio %', 'Current Liability to Current Assets', 'Cash Flow to Liability', 'Net Income to Total Assets' and 'Fixed Assets to Assets'. The sign of the monotonicity constraint was determined by consulting a book on corporate finance [3], and using common sense reasoning. For example, 'Net Income to Total Assets' is an indicator of a company's profitability. A higher ratio suggests that the company generates more profit relative to its total assets, indicating strong financial performance. Hence, it stands to reason that there is a monotonically decreasing relationship between this ratio and the probability that the company will go bankrupt. The signs for the other ratios were derived in a similar fashion.

Table 6: Classification datasets after preprocessing.

Name	N	number of features	proportion class 1
Bankrupt[9]	440	5	0.50
Compas[10]	7214	4	0.45
Credit[11]	592	5	0.50
Haberman[12]	306	3	0.26
Water[13]	1824	9	0.50

Table 7 displays the accuracy on the test set, for each constraint combination, and each training set size.

We witness the same phenomenon as with regression: the smaller the training set, the more prominent the effect of enforcing monotonicity constraints. For example, for the Bankruptcy data, the difference in accuracy between LG and $\bar{L}\bar{G}$ is 5 percentage points for a training set of size 50, but for size 150 both models perform equally well. Overall, it appears that enforcing both local and global monotonicity (LG) gives the best performance. It is only beaten once by another constraint configuration, and with only a minuscule difference. The accuracies obtained on Haberman are rather unimpressive accross the board, since predicting the majority class gives an accuracy of 74% already. This confirms that tree based models have problems with skewed class distributions, and have a tendency to prune back to the root node in such cases. On the Credit data all constraint combinations perform equally well, largely because a split on the single feature "Share" already leads to almost perfect separation of the classes.

[9] `https://www.kaggle.com/datasets/fedesoriano/company-bankruptcy-prediction`
[10] `https://www.kaggle.com/datasets/danofer/compass`
[11] `https://www.kaggle.com/datasets/dansbecker/aer-credit-card-data`
[12] `https://www.kaggle.com/datasets/gilsousa/habermans-survival-data-set`
[13] `https://www.kaggle.com/datasets/mssmartypants/water-quality`

Table 7: Test set accuracy for classification datasets.

	n	LG	$\bar{L}G$	$L\bar{G}$	$\bar{L}\bar{G}$
	50	0.77	0.75	0.76	0.72
Bankrupt	100	0.82	0.5	0.82	0.81
	150	0.82	0.82	0.82	0.82
	50	0.644	0.62	0.636	0.549
Compas	100	0.669	0.549	0.669	0.67
	150	0.64	0.64	0.6	0.62
	50	0.95	0.95	0.95	0.95
Credit	100	0.99	0.99	0.99	0.99
	150	0.98	0.98	0.98	0.98
	50	0.72	0.72	0.71	0.71
Haberman	100	0.757	0.757	0.699	0.748
	150	0.74	0.74	0.74	0.74
	50	0.71	0.5	0.66	0.57
Water	100	0.674	0.5	0.638	0.633
	150	0.77	0.77	0.77	0.77

Comparing the different constraint configurations statistically, we compute that for $n = 50$ the p-value of the Friedman test is 0.039, meaning that we reject H_0 at $\alpha = 0.05$. However, the p-values of the post-hoc Nemenyi test are all greater than α. For training set sizes 100 and 150, the p-values are 0.26 and 0.39 respectively, and therefore we cannot reject H_0.

5 Conclusion

We presented the first algorithm for learning globally monotone oblique decision trees. To determine if a given tree is globally monotone it suffices to verify the satisfiability of a set of linear constraints. If global monotonicity has been violated, the algorithm uses the isotonic regression to adjust the predictions in the leaf nodes.

Experiments on artificial and real data sets show that enforcing local and global monotonicity tends to improve predictive performance for relatively small training set sizes, but this advantage gets smaller and eventually disappears as the size of the training set grows. This makes sense: if the constraints are correct, then models trained on sufficiently large samples will already satisfy them. The experimental results also suggest that combining both constraints works better than each constraint on its own. Regardless of a possible improvement of predictive performance, enforcing the global constraint may be a requirement from the viewpoint of fairness, explanation, and model acceptance by the domain expert.

In future research, we will extend this work to ordinal classification problems with more than two classes. Also, we plan to simplify the unconstrained linear combination splits, e.g. through regularization, since currently these splits can become rather complex, possibly leading to overfitting.

References

1. C. Bartley, W. Liu, and M. Reynolds. Enhanced random forest algorithms for partially monotone ordinal classification. In *Proceedings of The Thirty-Third AAAI Conference on Artificial Intelligence*, pages 3224–3231. AAAI Press, 2019.
2. A. Ben-David. Monotonicity maintenance in information-theoretic machine learning algorithms. *Machine Learning*, 19:29–43, 1995.
3. J. Berk and P. DeMarzo. *Corporate Finance: Global Edition*. Pearson Education, 2011.
4. L. Breiman, J.H. Friedman, R.A. Olshen, and C.T. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, California, 1984.
5. J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.
6. B. Dutertre and L. Mendonça de Moura. A fast linear-arithmetic solver for DPLL(T). In T. Ball and R. B. Jones, editors, *Computer Aided Verification, 18th International Conference, CAV*, volume 4144 of *Lecture Notes in Computer Science*, pages 81–94. Springer, 2006.
7. M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32:675–701, 1937.
8. G. W. Ladd. Linear probability functions and discriminant functions. *Econometrica*, 34(4):873–885, 1966.
9. M. Magdon-Ismail and J. Sill. A linear fit gets the correct monotonicity directions. *Machine Learning*, 70(1):21–43, 2008.
10. J. Marques-Silva, T. Gerspacher, M.C. Cooper, A. Ignatiev, and N. Narodytska. Explanations for monotonic classifiers. In *International Conference on Machine Learning*, pages 7469–7479. PMLR, 2021.
11. M.J. Pazzani, S. Mani, and W.R. Shankle. Acceptance of rules generated by machine learning among medical experts. *Methods of Information in Medicine*, 40(5):380–385, 2001.
12. S. Pei and Q. Hu. Partially monotonic decision trees. *Information Science*, 424:104–117, 2018.
13. S. Pei, Q. Hu, and C. Chen. Multivariate decision trees with monotonicity constraints. *Knowledge Based Systems*, 112:14–25, 2016.
14. J. G. Propp and D. B. Wilson. Exact sampling with coupled markov chains and applications to statistical mechanics. *Random Structures & Algorithms*, 9(1-2):223–252, 1996.
15. T. Robertson, F. Wright, and R.L. Dykstra. *Order Restricted Statistical Inference*. Wiley, 1988.
16. J. Spouge, H. Wan, and W.J. Wilbur. Least squares isotonic regression in two dimensions. *Journal Of Optimization Theory And Applications*, 117(3):585–605, 2003.
17. R. van de Kamp, A. Feelders, and N. Barile. Isotonic classification trees. In N. Adams, editor, *Proceedings of IDA 2009*, volume 5772 of *LNCS*, pages 405–416. Springer, 2009.