Utrecht University

MASTER'S THESIS

Monotone Oblique Decision Trees

Author:
Maarten Al Sadawi
m.t.i.alsadawi@students.uu.nl

Program:
Computing Science

Supervisors:

Dr. Ad Feelders
Prof. Dr. Marc van Kreveld

July 7, 2023

Acknowledgement

I would like to express my gratitude to my supervisors, Dr. Ad Feelders and Prof. Dr. Marc van Kreveld, for their advice, feedback and guidance throughout my research. Special thanks to Dr. Ad Feelders, who served as my main supervisor and provided invaluable comments and support.

Abstract

In many data analysis applications, it is often reasonable to assume that the response variable increases or decreases in relation to one or more attributes or features. These relationships between the response and features are referred to as monotone. Monotonicity is not only plausible but can also be a desirable property of a decision model for the sake of explanation, justification, and fairness. Because the monotonicity constraint is common in practice, several data analysis techniques have been adapted to accommodate such constraints. For instance, various algorithms have been developed for learning monotone decision trees. However, these algorithms only consider splits on single attributes, resulting in axis-parallel splits and a partitioning of the attribute space into rectangular areas. In contrast, oblique decision trees allow for linear combination splits like $w_1x_1 + w_2x_2 > c$.

To the best of our knowledge, no existing algorithms enforce monotonicity for this more complex partitioning of the attribute space. We have developed such an algorithm and evaluated its performance (MSE or accuracy) through experiments using well-known datasets for monotone classification and regression, as well as artificially constructed data.

Appealingly, enforcing monotonicity constraints is beneficial to enhance performance in both classification and regression tasks. Furthermore, the smaller the train set is, the more prominent the effect is of enforcing monotonicity constraints, as the performance gap between the algorithm mode that enforces monotonicity compared to its unconstrained counterpart tends to decline on the test sets.

Moreover, when dealing with artificial data, we discovered that all else equal, accuracy tends to significantly decline with the number of features. This effect is most emphatic with respect to the zero and negative datasets. Furthermore, all else equal, accuracy tends to significantly increase with correlation between the features.

Thus, assuming the underlying nature of the data adheres to monotone relationships between features and the response variable, enforcing monotonicity constraints is advantageous.

Contents

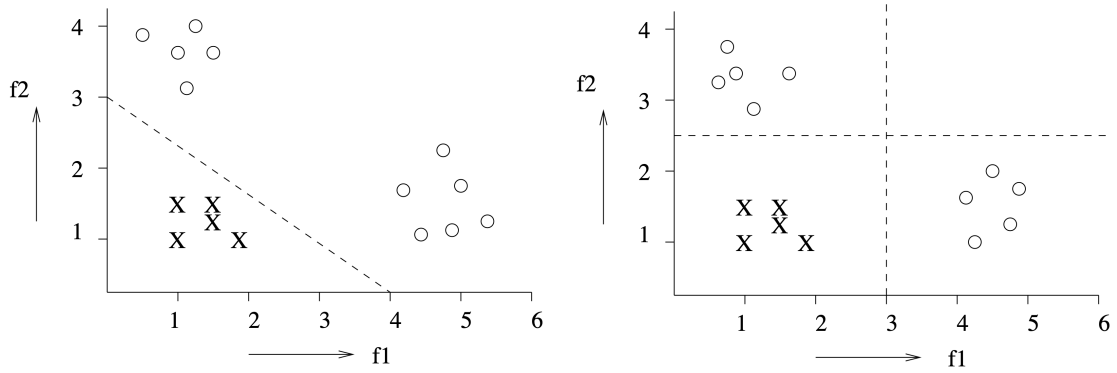
Chapter 1

Introduction and problem statement

In many data analysis applications, it is reasonable to assume that the response variable, also known as the target variable, increases or decreases in relation to one or more features. These relationships are referred to as monotone relations. Given the common occurrence of monotonicity constraints, various data analysis techniques have been adapted to handle them. Examples of such techniques include classification and regression trees, nearest neighbor algorithms, neural networks, and support vector machines [1]–[4].

Monotonicity is not only plausible but also a desirable property in decision models for the purpose of explanation, justification, and fairness. For instance, banks use monotonicity when making acceptance or rejection decisions on loan applications [3]. In the context of house pricing, it has been observed that the selling price of a house generally increases with factors like the size of the lot and the number of rooms, while decreasing with factors such as crime rate or pollution concentration in the area [5]. Additionally, research has demonstrated that rules learned with monotonicity constraints are more readily accepted by medical experts compared to rules learned without such constraints [6].

Oblique decision trees offer the potential for more compact models compared to decision trees that only allow axis-parallel splits. This is because oblique trees enable more complex splitting rules and, consequently, a more intricate partitioning of the feature space. Figure 1.1 illustrates this concept, where the oblique decision tree achieves a more compact structure with only two leaves instead of three.



(a) Decision tree with an oblique split. (b) Decision tree without oblique split.

Figure 1.1: Two examples of decision trees where there are two features f1, f2 and two class labels: a circle and a cross. Images are from [7].

Algorithms for oblique decision trees already exist [8]. As far as our knowledge goes, however, there is currently no algorithm available for generating a *globally monotone* oblique decision tree. By globally monotone, we mean that the resulting decision tree always satisfies the monotonicity constraint. Since oblique decision trees can offer more compact models compared to non-oblique ones, it is desirable to enforce this constraint on these types of trees as well.

The objective of this research is to develop an algorithm such that the monotonicity constraint with respect to oblique decision trees is always satisfied. We aim to evaluate this algorithm through experiments conducted on well-known datasets for monotone classification and regression tasks.

In this thesis, our objective is to address the following research questions:

1. How can one verify whether a given oblique classification/regression tree is monotone?

2. How can one develop a justified approach to construct a monotone oblique classification/regression tree?

3. How does the predictive performance of a monotone oblique classification/regression tree compare to that of its unconstrained counterpart?

This thesis is structured as follows. The upcoming chapter will introduce the fundamental concepts and notation that will be utilised throughout the thesis. Chapter 3 will encompass an exploration of related works pertaining to monotone/oblique decision trees. The subsequent chapter, chapter 4, will delve into the developed algorithm and the specific design choices made. In chapter 5, a comprehensive overview of the utilised datasets and their selected features will

be presented, along with the primary findings resulting from evaluating the algorithm. The thesis will conclude in chapter 6, summarising significant discoveries and providing avenues for future research. The appendices will furnish additional details on the algorithm, datasets, and evaluation tables incorporating multiple metrics. Finally, `https://github.com/ZazeyManda/IDT-oblique.git` contains our complete code and used datasets.

Chapter 2

Preliminaries

2.1 Background

Different classification and regression models are helping the task of decision-making, such as linear/logistic regression or artificial neural networks. This research however will focus on *classification and regression trees*. A classification tree (also known as a decision tree) is a predictive model (classifier) in the field of data mining which uses a supervised learning approach. With supervised learning the model is trained using *labelled* examples from a dataset. The label is also known as the class of the example. Thus on unseen data (where the class is unknown), the classifier will make an educated guess on its class label. For example, good/bad credit for loan applicants, spam/no spam for e-mail messages or the numbers 0 through 9 for handwritten digits [9]. The first two are associated with binary classifiers as they only have two output classes. The last one is associated with a multiclass classifier. An example binary classification tree built from table 2.1 is found in figure 2.1. Consider a new applicant with the following information; age: 44, married: yes, own house: yes, income: 30,000, gender: male. We follow along the appropriate splits (starting from the root node) until we reach a leaf node. At this point, the leaf node predicts the majority class (that is, the most common class of the data points in the leaf). So for our example applicant we will end up in the second leaf node, thus predicting class 'good' credit for a loan. The construction of such a tree from the data is also discussed in [9], but the main idea is that we try to aim for leaf purity using some impurity function (like Gini) to determine the best split for each node. With leaf purity, we mean that leaves contain only examples of the data with the same class. In the case of regression trees, we predict a number such as the sale prices of houses in a neighbourhood. The impurity function is in that case the residual sum of squares (RSS). The formula is

$$RSS = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2.1}$$

where n is the amount of data points in a node, y_i is the value of the target variable of data point i and $\hat{y}_i$ is the predicted value of the target variable of data point i. In the case of sale prices of houses, the target variable is not a class label but a continuous number starting from \$0 (in reality no house will probably be sold for \$0). Suppose further that a node contains the following values for the target: 12,16,19 and 45. The mean is 23. The residual sum of squares equals

$$(12 - 23)^2 + (16 - 23)^2 + (19 - 23)^2 + (45 - 23)^2.$$

This is the impurity of the node with respect to the target.

One interesting point to note about Gini impurity is that it is the variance of a Bernoulli random variable. If p is the relative frequency of class label 1 at some node, then its Gini impurity is $p(1 - p)$ (in the case of binary class labels). Thus when splitting, we try to minimise the variance of the class label within a node; consequently making a node purer. Note further that $\hat{y}_i$ of equation 2.1 is the prediction of the node, in case of regression there is no majority class and thus this is equal to the mean of the target variable of the data points in the node. Thus, if we divide the equation by n, we would get the mean squared error. This corresponds directly to the variance around the fitted regression line. Thus both impurity functions try to minimise the variance of the target variable within a node.

Since regression tasks are analogues to classification tasks, the examples in the remainder of this chapter are only given for classification.

Record	age	married	own house	income	gender	class
1	22	no	no	28,000	male	bad
2	46	no	yes	32,000	female	bad
3	24	yes	yes	24,000	male	bad
4	25	no	no	27,000	male	bad
5	29	yes	yes	32,000	female	bad
6	45	yes	yes	30,000	female	good
7	63	yes	yes	58,000	male	good
8	36	yes	no	52,000	male	good
9	23	no	yes	40,000	female	good
10	50	yes	yes	28,000	female	good

Table 2.1: Example of bank credit data [9].

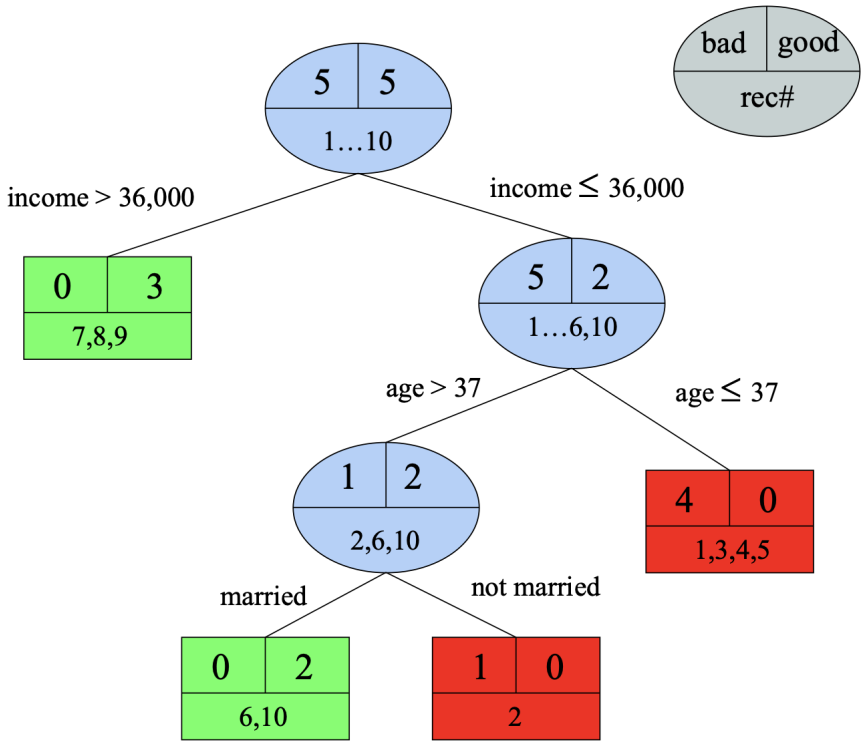


Figure 2.1: Classification tree built on credit scoring data [9]. Here, rec# stands for the record numbers in table 2.1

We have seen an example of a binary classification tree in figure 2.1. With binary, we mean that the splits are binary and that the class label is binary. A split of this tree is one of the following forms:

- $x \leq c$,
- $x \in X$,

where x is a feature/attribute (for example income or gender), c is a constant and X is a set in the case that x is a variable taking categorical values. In regular trees, a split happens only on a single feature at a time. This gives rise to axis-parallel splits and partitioning of the feature space into rectangular areas. The problem with that is for example if the true decision boundary is not parallel to an axis but some linear function, a tree would resort to potentially many axis-aligned splits instead of a linear combination (of features) split. Hence, oblique trees have been developed [8] which allow linear combination splits of the form $\mathbf{w^T x} \leq c$ where $\mathbf{x} = (x_1, x_2, \cdot, x_p)$ is the feature-vector and $\mathbf{w} = (w_1, w_2, \cdot, w_p)$ the weights/coefficients-vector; c is again a constant and p is the number of features. Note that this is a generalisation to an axis parallel split on a numeric feature. One

possible way to construct these oblique trees is to apply repeated logistic regression on the data since logistic regression gives exactly such a linear combination decision boundary.

2.2 Concepts

In this section, the concepts mentioned in section 2.1 are formally described and explained. Throughout the remainder of this thesis, we assume there are p features (unless otherwise stated), and we define the feature space $\mathcal{X}$ to be $\mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_p$ where $\mathcal{X}_i$ is the domain of feature i ($1 \leq i \leq p$). We assume that $\mathcal{X}$ consists of only numerical features. A feature is a numerical feature if its domain is a set of numbers that are in an ordered relationship, for example, $\{\mathbb{R}\}$. In the remainder we use the word *polytope*, with this we mean a geometric object with flat sides (faces) in $\mathcal{X}$. We would like to note that polytopes are a generalisation of polyhedra in three dimensions, which in its turn is a generalisation of polygons in two dimensions. A convex polytope is a polytope that is also a convex set in $\mathcal{X}$. This means that a line segment between any two points in a polytope t also belongs to t (all points of the line segment belong to t). A convex polytope can also be defined as an intersection of a finite number of half-spaces, the so-called H-representation. A half-space is specified via a linear inequality $a_1x_1 + a_2x_2 + \cdots + a_nx_n \leq b$ where $a_1, \ldots, a_n, b \in \mathbb{R}$. This half-space is called closed as there is no strict inequality, if we were to use $<$ the half-space would be called open. For our purposes, it does not matter if a polytope is bounded or unbounded. We now define a closed convex polytope.

Definition 2.2.1 (Closed convex polytope)**.** *A closed convex polytope is as a set of solutions to a system of linear inequalities. That is, all* $\mathbf{x}$ *that satisfy* $A\mathbf{x} \leq \mathbf{b}$ *where* A *is a* $m \times n$ *matrix of real-valued coefficients,* $\mathbf{x}$ *is a* $n \times 1$ *column vector whose coordinates are the variables* x_1 *to* x_n *and* $\mathbf{b}$ *is a* $m \times 1$ *column vector whose coordinates are the right-hand side of the linear inequalities.*

The intersection of the splits on the path from the root node to a leaf node in the oblique tree forms a convex polytope. This is not necessarily a fully closed convex polytope because the splits could contain strict inequalities. We say that a boundary of a convex polytope is open if it does not belong to that polytope.

It is important to note that features in the feature space could take on real values, integer values or even binary values. Furthermore, the features do not need to be bounded.

2.2.1 Monotone prediction

The classification task is to build a classification/regression function $f : \mathcal{X} \to \mathcal{C}$ that predicts a value $c \in \mathcal{C}$ given an input point $\mathbf{x} = (x_1, x_2, \ldots, x_p)$, where $\mathcal{X}$ is the feature space and p is the number of features. In the case of ordinal classification, $\mathcal{C} = \{c_1, c_2, \ldots c_C\}$ consists of C classes such that there is a total order on them: $c_1 \leq c_2 \leq \cdots \leq c_C$. However, the distance between classes is *undefined*. In our context, a monotone ordinal classifier has a non-decreasing (non-increasing) impact on the output class (response variable) for every feature in $\mathcal{X}$. Furthermore, we assume that each feature i in $\mathcal{X}$ takes values x_i in a linearly ordered set $\mathcal{X}_i$. In the case of regression, $\mathcal{C}$ is not a set of labels but usually a number in $\mathbb{N}$, $\mathbb{Z}$ or $\mathbb{R}$. In the remaining parts of our thesis, we use $\mathcal{C}$ in the context of classification as well as regression. We will now define the product order $\preceq$ on $\mathcal{X}$.

Definition 2.2.2 (Product order)**.** *Let $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ where $\mathbf{x} = (x_1, x_2, \ldots, x_p)$ and $\mathbf{x}' = (x'_1, x'_2, \ldots, x'_p)$. We define the product order $\preceq$ on $\mathcal{X}$ as $\mathbf{x} \preceq \mathbf{x}'$ if and only if $x_i \leq x'_i$ for all $1 \leq i \leq p$.*

Note that our product order is a partial ordering on $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_p$. The assumption that $\mathcal{X}$ solely includes numerical features is important here.

A monotone ordinal classification or regression function is a function $f : \mathcal{X} \to \mathcal{C}$ for which

$$\mathbf{x} \preceq \mathbf{x}' \implies f(\mathbf{x}) \leq f(\mathbf{x}') \tag{2.2}$$

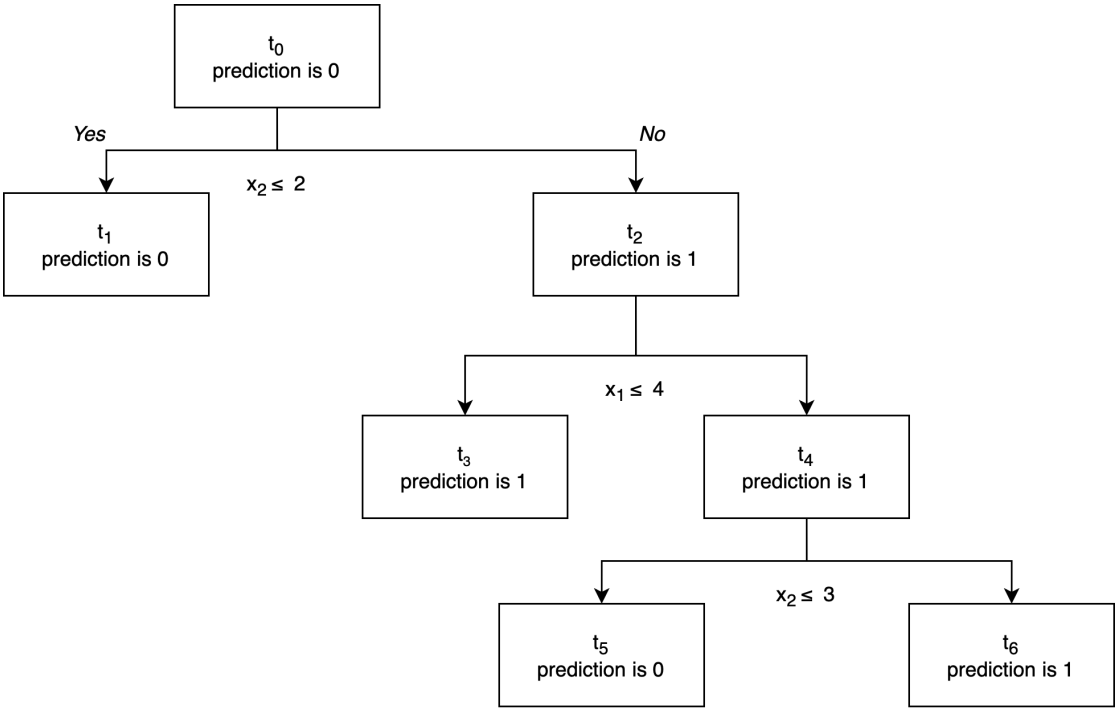
for all instances $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$. This type of monotone relation is also known as *isotonic*.

Definition 2.2.3 (Monotone dataset)**.** *A dataset $\{\mathbf{x}_i, c_i\}_{i=1}^n$ is called monotone if for all i, j we have $\mathbf{x}_i \preceq \mathbf{x}_j \implies c_i \leq c_j$.*

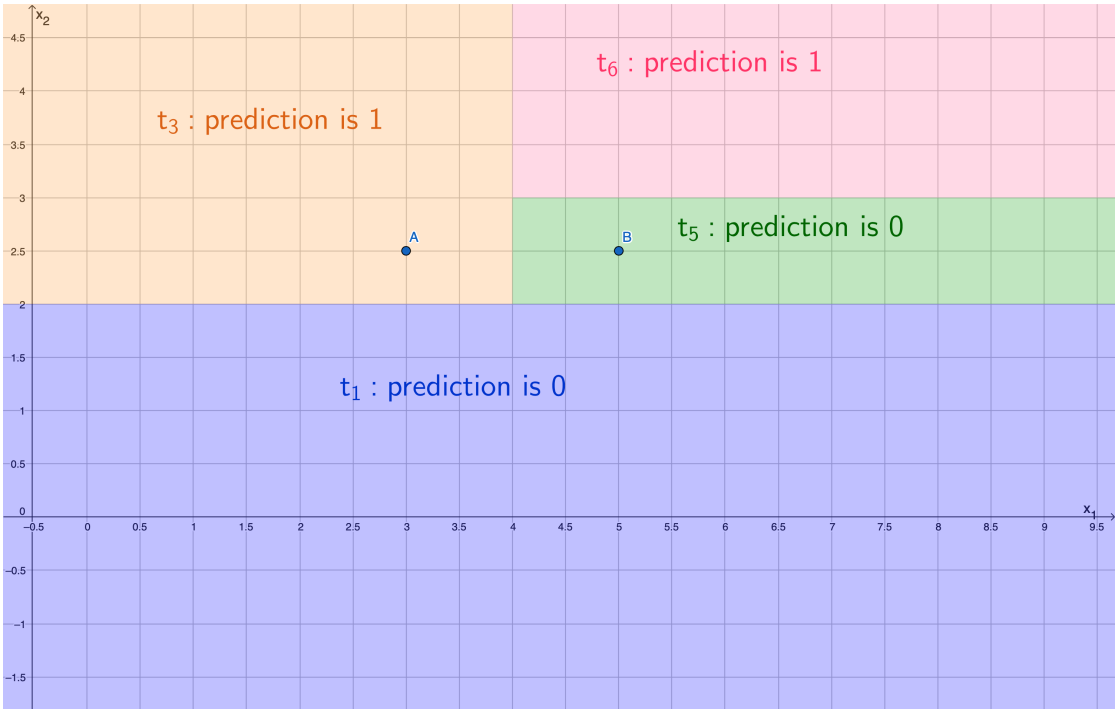
Local vs. global monotonicity We shall now return to the context of binary classification trees with binary class labels. In this paragraph, we say that a data point is predicted to be 1 if it falls in a leaf whose relative frequency of class label 1 is greater than or equal to $\frac{1}{2}$ and 0 otherwise. That is, we are rounding the prediction to the nearest integer (0 or 1). The leaf prediction of some leaf ℓ is calculated as the sum of all class labels of all data points inside ℓ, divided by the total number of data points in ℓ.

There are two paradigms to ensure monotonicity for a tree: local or global. Global monotonicity implies that the tree *always* satisfies equation 2.2. Local monotonicity on the other hand can only guarantee that the splits are monotone. This means that for some parent node, if a splitting rule has the form $x_i \leq c$, then data points that comply with the rule go to a child node that does not predict a higher class label than data points that do not comply with the rule, for

some feature x_i and constant c. That this does not guarantee global monotonicity becomes clear in figure 2.2a. Here, the classes are either 0 or 1, t_0 is the root node and we only have two features. The tree is locally monotone but not globally due to leaf t_5. For example given $\mathbf{A} = (3, 2\frac{1}{2})$ and $\mathbf{B} = (5, 2\frac{1}{2})$, we have that $\mathbf{A} \preceq \mathbf{B}$ but $f(\mathbf{A}) > f(\mathbf{B})$, as can be seen in figure 2.2b.



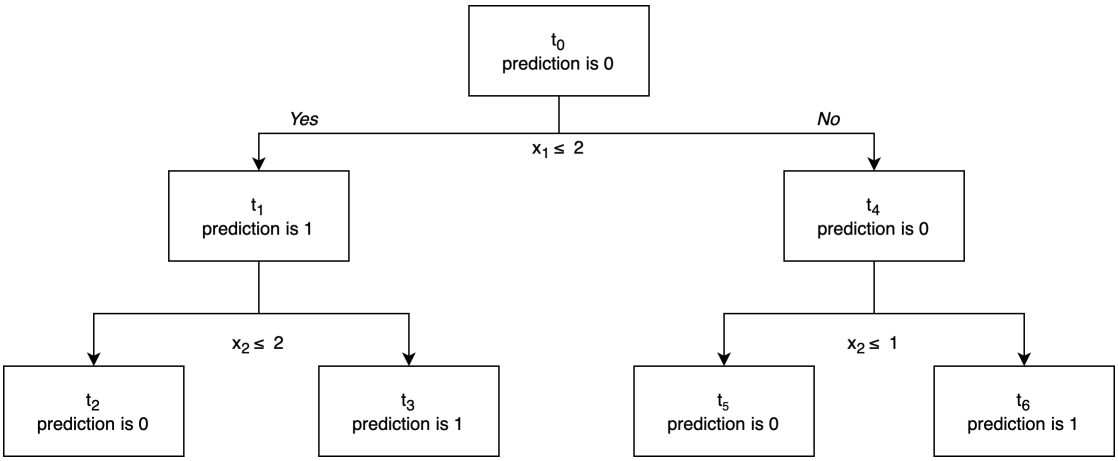
(a) A classification tree.



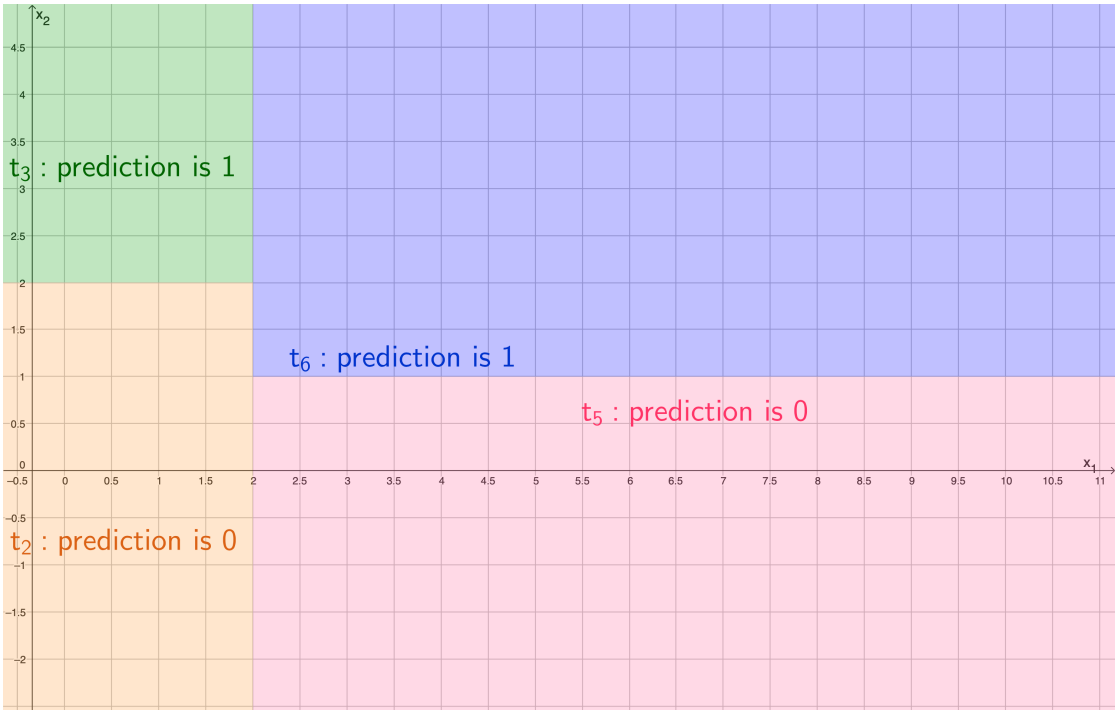
(b) Leaves of tree in figure 2.2a visualised.

Figure 2.2: Example of classification tree with two numerical features, and the leaves visualised. The tree is locally but not globally monotone.

On the other hand, a tree that is globally monotone does not need to be locally monotone. An example of such a tree is found in figure 2.3. We have that t_1, t_4 are children of t_0 and the data points in t_1 comply with the first split, yet t_1 predicts a higher class label than t_4.



(a) A classification tree.



(b) Leaves of tree in figure 2.3a visualised.

Figure 2.3: Example of a tree with two numerical features, and the leaves visualised. The tree is globally but not locally monotone.

We introduce three additional definitions with respect to point and leaf domination.

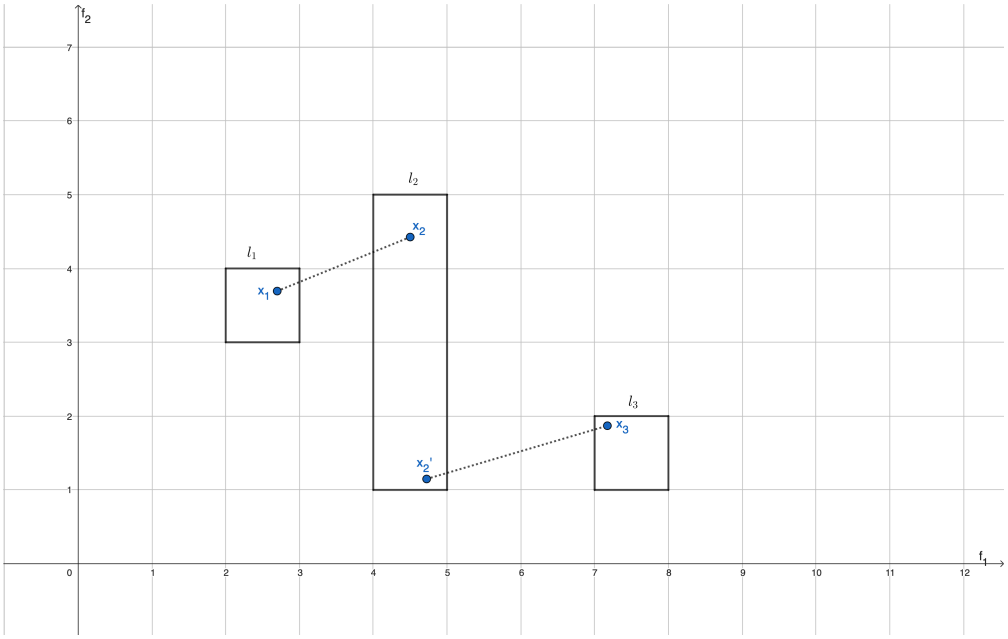
Definition 2.2.4 (Point domination)**.** *Let* $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$*. We say that* $\mathbf{x}'$ *dominates* $\mathbf{x}$ *if and only if* $\mathbf{x} \preceq \mathbf{x}'$*.*

Definition 2.2.5 (Leaf domination)**.** *Let* t, t' *be convex polytopes (leaves) in* $\mathcal{X}$*. We say that* t' *dominates* t *if there exists a point in the closed part of* t' *that dominates a point in the closed part of* t*, written as* $t \preceq t'$*. If both leaves dominate each other, we say* $t = t'$*.*

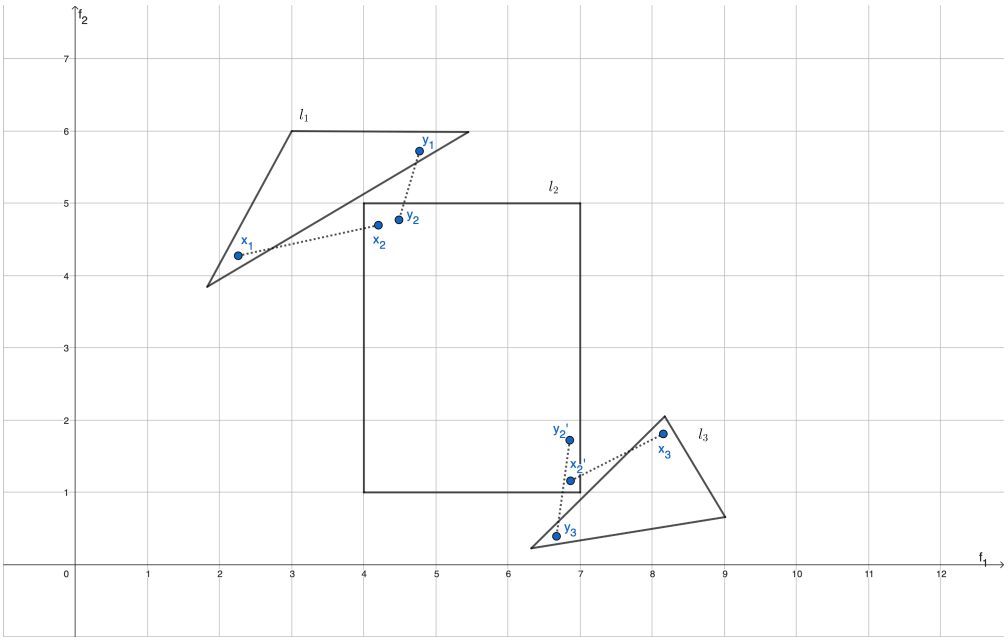
Definition 2.2.6 (Incomparable points)**.** *Let* $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$*. We say that* $\mathbf{x}$ *and* $\mathbf{x}'$ *are incomparable, written as* $\mathbf{x} \sim \mathbf{x}'$*, if and only if there is no dominance relation between them. That is, there exists* i, j *such that* $x_i > x'_i$ *and* $x_j < x'_j$*.*

Definition 2.2.7 (Incomparable leaves)**.** *Let* t, t' *be convex polytopes (leaves) in* $\mathcal{X}$ *We say that* t *and* t' *are incomparable, written as* $t \sim t'$*, if and only if there is no dominance relation between them.*

Leaf order With these definitions, we are able to talk about an ordering on the leaves. Leaf t precedes leaf t' in the order if $t \preceq t'$. If $t = t'$, then they share an equal position in the ordering. The ordering between t and t' is not defined in the case of $t \sim t'$. For simplicity, this ordering on the leaves is called *leaf order*. To gain a better understanding of the leaf order, see figure 2.4. In this figure, we present two examples to demonstrate the lack of transitivity in our leaf order. Figure 2.4a shows that leaf $l_1 \preceq l_2$ and $l_2 \preceq l_3$ but $l_1 \sim l_3$. Figure 2.4b shows that mutual dominance is also not transitive, since $l_1 = l_2$ and $l_2 = l_3$ but $l_1 \sim l_3$.



(a) Example of three leaves of some tree, assuming we have two features f_1 and f_2. We have $\mathbf{x}_1 \preceq \mathbf{x}_2$ and $\mathbf{x}'_2 \preceq \mathbf{x}_3$.



(b) Example of three leaves of some tree, assuming we have two features f_1 and f_2. We have $\mathbf{x}_1 \preceq \mathbf{x}_2$, $\mathbf{y}_2 \preceq \mathbf{y}_1$, $\mathbf{x}'_2 \preceq \mathbf{x}_3$ and $\mathbf{y}_3 \preceq \mathbf{y}'_2$.

Figure 2.4: Here are two examples of three leaves, with dashed lines indicating point domination between points in different leaves.

As the convex polytopes resulting from the splitting of oblique trees are not necessarily closed, an open boundary of a polytope does not belong to that polytope. More specifically, there could be a point on an open boundary of some polytope t that dominates a point on a boundary of another polytope t' such that all points in t are incomparable with all points in t'. See for example figure 2.5 where this is the case. Whether such polytopes (leaves) are constructed by a tree depends on the allowed splitting rules.

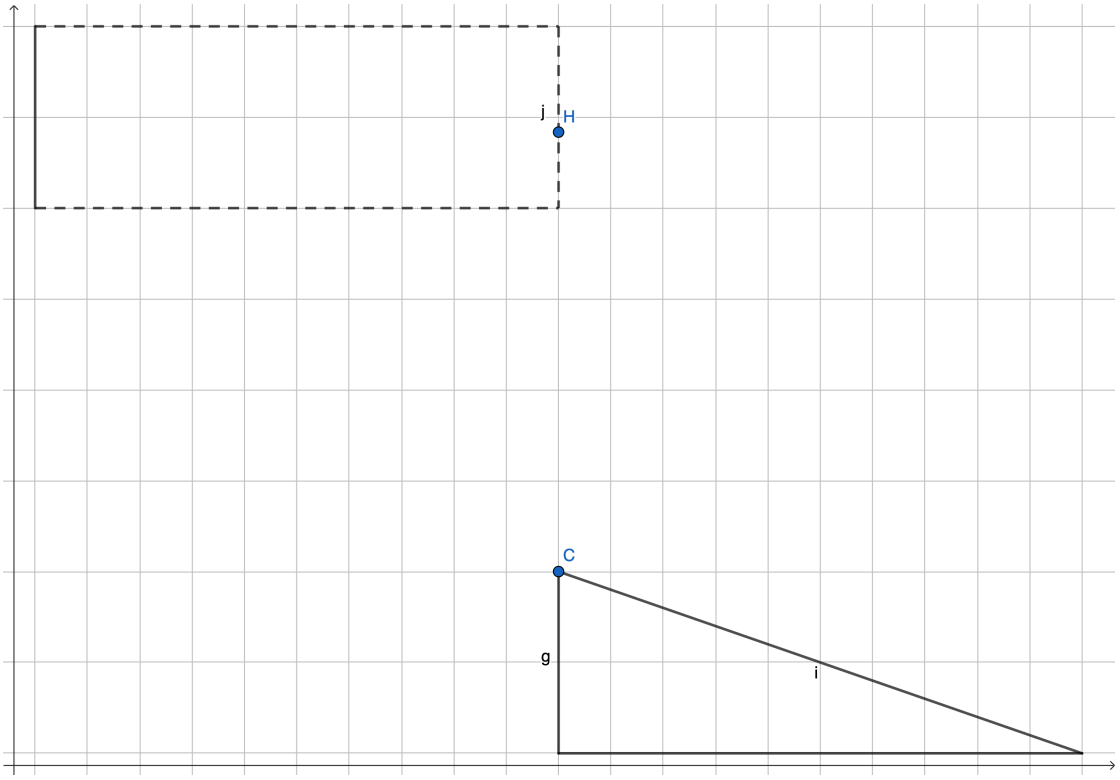


Figure 2.5: Here, a dashed line segment denotes an open boundary. Point H on the open boundary j of the rectangle dominates point C on the boundaries g,i of the triangle. But, no point in the rectangle dominates a point of the triangle.

2.2.2 Partial monotone classification or regression

If we allow categorical features, the feature space is extended from $\mathcal{X}$ to $\mathcal{X} \times \mathcal{Z}$ where $\mathcal{Z} = \mathcal{Z}_1 \times \mathcal{Z}_2 \times \ldots \mathcal{Z}_q$ and $\mathcal{Z}_i$ is the domain of categorical feature i $(1 \leq i \leq q)$. There are q categorical features. The domain of a categorical feature is a set of discrete values that are not in an ordered relationship. A partial monotone classification or regression function is defined as a function $f : \mathcal{X} \times \mathcal{Z} \to \mathcal{C}$ for

which

$$\mathbf{x} \preceq \mathbf{x}' \wedge \mathbf{z} = \mathbf{z}' \implies f(\mathbf{x}, \mathbf{z}) \leq f(\mathbf{x}', \mathbf{z}) \tag{2.3}$$

for all instances $(\mathbf{x}, \mathbf{z}), (\mathbf{x}', \mathbf{z}') \in \mathcal{X} \times \mathcal{Z}$.

In the remaining parts of this thesis, however, we use (regular) monotone classification/regression for the sake of simplicity and only mention partial monotone classification/regression if it becomes relevant within the given context.

2.2.3 Pruning trees

In general, tree algorithms grow the tree by continuing splitting until all leaf nodes all pure i.e., contain examples of a single class. This gives rise to the problem of overfitting. Two solutions are to either prune the tree (merge child nodes back to parent node) or include stopping rules (no node expansion if the impurity reduction of the best split is below some threshold) [9]. The main disadvantage of using stopping rules is that sometimes a 'weak' split (achieving less impurity reduction) must be made to follow up with a 'strong' split (achieving higher impurity reduction). Pruning can be done in various ways but one of the more popular techniques is called cost complexity pruning [10].

Chapter 3

Related work

In this chapter, we review related work on algorithms that opt to enforce monotonicity on a classification tree, as well as algorithms that provide techniques to construct an oblique classification tree.

3.1 Isotonic classification trees

The isotonic classification tree (ICT) algorithm is introduced by Van de Kamp et al. [1]. They discussed it in the context of regular binary classification trees which allow for more than two class labels. The algorithm adjusts the leaf predictions (also called relabelling) in such a way that the tree guarantees global monotonicity. To determine whether a given tree is monotone, consider the minimum and maximum element of some leaf t, respectively t'. The claim is that if $\min(t) \prec \max(t')$, then node t contains points that are smaller than some points in the node t', so the class label of leaf t should not be greater than the class label of leaf t'. In case of violation of this requirement, the class labels of the leaf nodes are repaired via *isotonic regression*. Van de Kamp et al. do not adjust the splitting conditions of a tree in their work. Their experiments have shown that ICT usually has a lower error on real-life datasets compared to standard trees, while also producing smaller trees.

3.2 Monotone rule random forest

Bartley et al. [2] claimed that the monotone classification tree produced by the ICT algorithm has high loss/bias. Since high bias cannot be corrected in hindsight, accuracy suffers. For that reason, the monotone rule random forest (MR-RF) algorithm was developed. We note that the proposed algorithm was developed for random forests, as opposed to the earlier discussed ICT which was developed for a

single tree. The authors used -1 and 1 as class labels in the case of binary classification. MR-RF enforces global monotonicity by adjusting the splitting conditions of the trees. Since their context is binary classification using random forest, the final prediction, given a feature vector $\mathbf{x}$ is calculated as follows:

$$sign\left(\sum_{t=1}^{T}\sum_{l=1}^{L_t} a_{t,l} f_{t,l}(\mathbf{x})\right) \tag{3.1}$$

where $f_{t,l}(\mathbf{x}) = 1$ if $\mathbf{x} \in$ leaf l of tree t and 0 otherwise and

$$a_{t,l} = \frac{1}{N}\sum_{i=1}^{N} c_i \frac{b_{i,t,l}}{K_{t,l}} \tag{3.2}$$

is the leaf prediction, such that $c_i \in \{-1, 1\}$, $b_{i,t,l}$ is the number of occurrences of data point $\mathbf{x}_i$ in the bootstrap sample in leaf l of tree t, $K_{t,l}$ is the total number of bootstrap data points in leaf l of tree t and T, N, L_t are the number of trees, training points and leaves in tree t respectively. The $c_i \frac{b_{i,t,l}}{K_{t,l}}$ in the sum of equation 3.2 is the contribution of the i'th data point in the training set to the proportion of the positive/negative cases in leaf l of tree t. For example, suppose that for a fixed i the point $\mathbf{x}_i$ occurs twice in the bootstrap sample of tree t, and $\mathbf{x}_i$ ends up in some leaf l of tree t, and $c_i = -1$, $K_{t,l} = 100$. We then have that

$$c_i \frac{b_{i,t,l}}{K_{t,l}} = -1 \cdot \frac{2}{100} = -\frac{1}{50} \tag{3.3}$$

It follows that equation 3.2 can be interpreted as the proportion of positive cases minus the proportion of negative cases in leaf l of tree t. This means that the leaf prediction on itself is not necessarily -1 or 1. Rather, the individual leaf prediction is a number in the interval $[-1, 1]$. Therefore, Bartley et al. distinguish whether a leaf prediction is positive or negative (≤ 0).

For a single tree, consider the path from the root node to some leaf node ℓ. Suppose that monotonicity needs to hold for all features. (The features where this does not need to hold for are simply ignored). Denote with $R_\ell(\mathbf{x}) = r_1(\mathbf{x}) \wedge \cdots \wedge r_\ell(\mathbf{x})$ the conjunction of splitting conditions from the root node to a leaf node ℓ, where $r_k(\mathbf{x}) \in \{x_i \leq u_k, x_i > l_k\}$ for some real values u_k, l_k and where x_i is the value of feature i of data point $\mathbf{x}$. If the leaf prediction of ℓ is positive, MR-RF will *transform* the leaf (conjunctions) as $R_\ell(\mathbf{x}) = \mathbf{x} \succ \mathbf{x}_\ell$, where $\mathbf{x}_\ell = (x_1, x_2, \ldots, x_p)$ is constructed for each leaf ℓ as follows:

$$x_i = \max[\{-\infty \cup \{l_k \mid 1 \leq k \leq \ell\}\}]. \tag{3.4}$$

Otherwise, if the leaf prediction is negative, the transformation is going to be $R_\ell(\mathbf{x}) = \mathbf{x} \prec \mathbf{x}_\ell$, where

$$x_\ell^i = \min[\{\infty \cup \{u_k \mid 1 \leq k \leq \ell\}\}]. \tag{3.5}$$

Note, the leaf ℓ after this transformation does not necessarily have the same shape in the feature space as the original leaf. If all splitting conditions leading to leaf nodes are changed like this, then the classification tree is globally monotone. Consider for example the classification tree in figure 3.1.

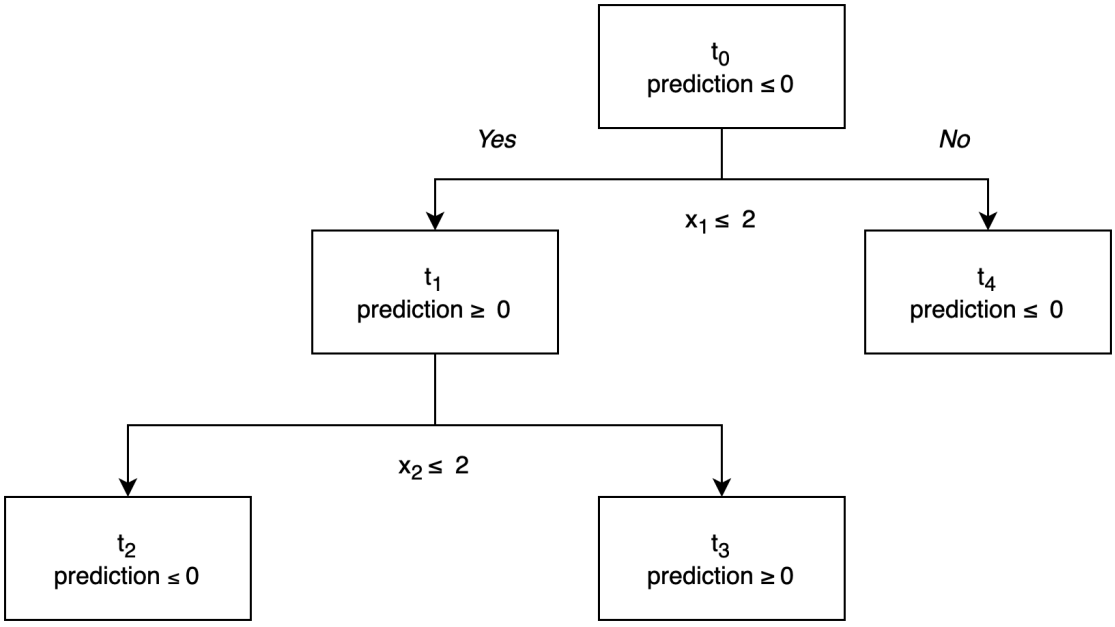
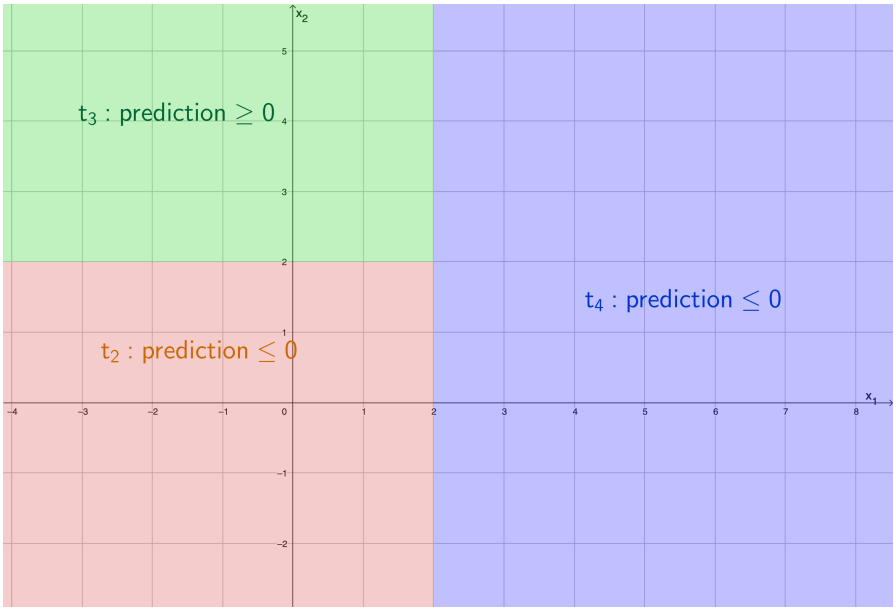
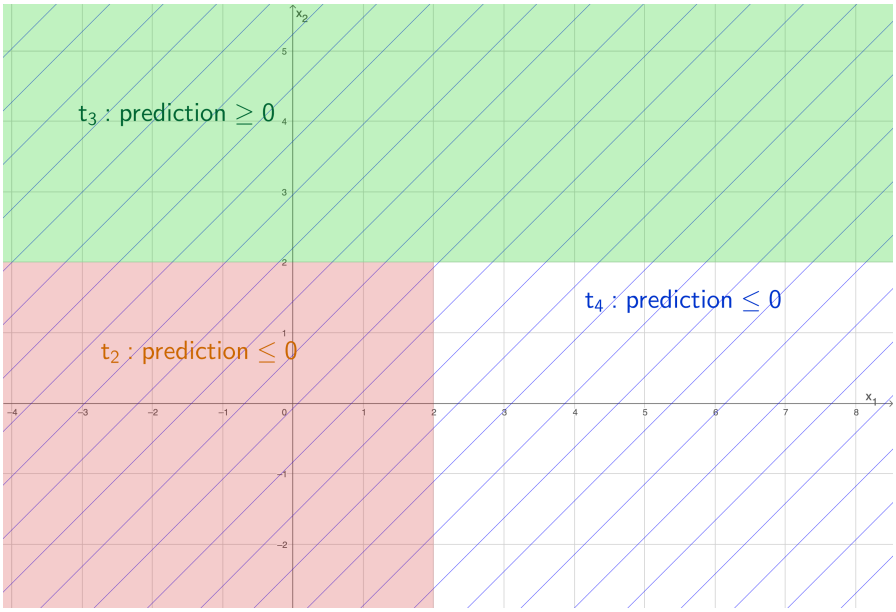


Figure 3.1: Example of binary classification tree that is neither globally nor locally monotone.

See figures 3.2a and 3.2b as an example of the effect on the leaves after transformation using MR-RF on the leaves of this tree.



(a)



(b)

Figure 3.2: Subfigure (a) is the visualisation of the leaves of the tree in figure 3.1. Subfigure (b) is the visualisation after applying the MR-RF transformation. Note that leaf t_4 comprises the entire feature space.

Consider leaf t_3 of figure 3.2 as an example. The splits leading to this leaf are $x_1 \leq 2, x_2 > 2$. Since the leaf prediction is positive, the base point $x_{t_3} = (-\infty, 2)$.

Thus leaf t_3 is after transformation represented by its base point x_{t_3}: all data points greater than x_{t_3} will fall into leaf t_3.

However, the leaves may now overlap and they do not necessarily contribute equally to the final prediction of some feature vector $\mathbf{x}$. The algorithm proposes two solutions that re-calculate leaf predictions: constrained logistic regression and naive Bayesian approximation.

Bartley et al. claimed that MR-RF is the state-of-the-art monotone tree ensemble in terms of performance (speed and accuracy). More specifically, their hypothesis is: MR-RF achieves lower loss monotonisation (bias) and higher accuracy, compared to other globally/locally monotone random forest algorithms. So Bartley et al. changed regular monotone classification tree algorithms such as ICT to work in a random forest setting. Their experiments are based on 17 datasets from UCI and KEEL repositories, taken from medical and finance fields.

Both variants of MR-RF (constrained and Bayesian) outperformed various other globally and locally monotone algorithms in three different accuracy metrics: kappa, F_1 score and Mean Absolute Error (MAE) score. They used Hommel's significance testing on the rank differences of these accuracy metrics with a significance level of $\alpha = 0.05$. Both variants however were not significantly more accurate than the ICT algorithm based on the MAE score. In terms of speed (complexity), the constrained variant turns out to be quite slow whereas the Bayesian variant is highly competitive in predictive performance and speed.

3.3 Multivariate decision trees with monotonicity constraints

Pei et al. [11] claim (without proof) an algorithm that guarantees local monotone splits in oblique trees, only if a dataset itself is monotone. A dataset is monotone if for two points $\mathbf{x}, \mathbf{y}$ in the set we have $\mathbf{x} \preceq \mathbf{y}$, then the class label of point $\mathbf{y}$ is at least as high as of point $\mathbf{x}$. They denote the splitting conditions for some input $\mathbf{x}$ as linear functions of the form $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$, for some weight vector $\mathbf{w}$ and bias b. These linear functions are constructed within each node via a variant of linear regression known as the non-negative least squares method (applied to the data points of that splitting node). A reason for doubt whether this technique gives the desired local monotone split is illustrated in figure 3.3. The dataset consisting of four points is monotone, yet it could be that a split is constructed which is not locally monotone.

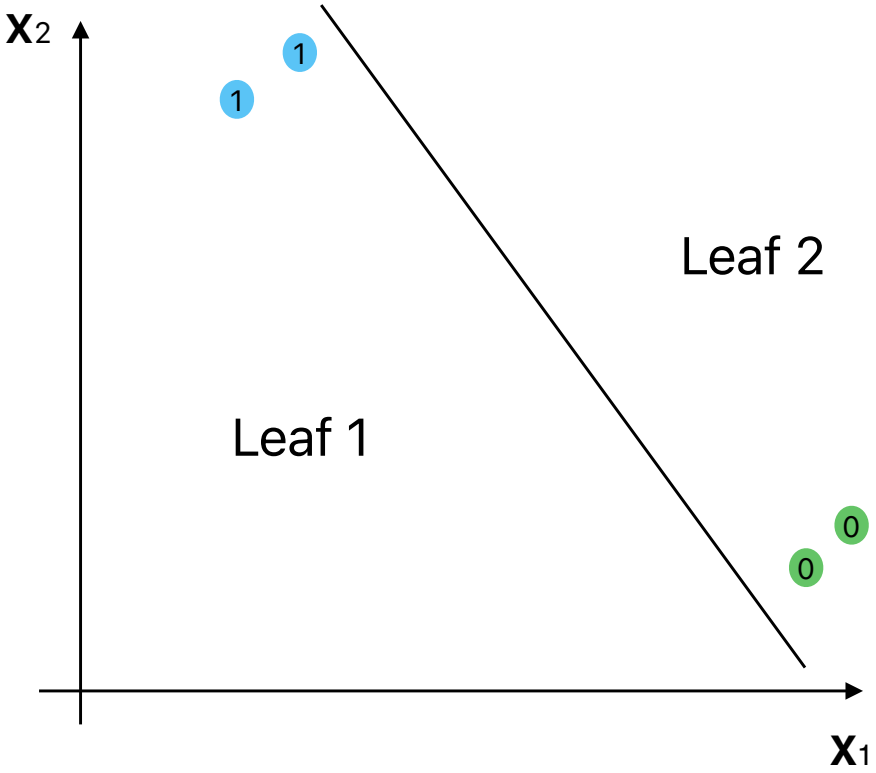


Figure 3.3: Non-local monotone split with non-negative coefficients.

Furthermore, if the dataset is not monotone, local monotonicity does not need to hold. For example, consider a dataset which contains points $\mathbf{x}, \mathbf{y}$ where $\mathbf{x} \preceq \mathbf{y}$. An oblique split constructed by the non-negative least squares method does not imply that $\mathbf{y}$ will end up in a child node whose class label is at least as high as that of $\mathbf{x}$. That depends on the labelling in the dataset.

In the conclusion of [11], the authors mention the challenging task of developing a globally monotone classification algorithm, which is left open for future research. Our objective is to precisely address this task by constructing such an algorithm. However, in our approach, we adopt the non-negative least squares method when creating splits in our oblique trees. If we do not do this, we could get unfavourable splits as can be seen in figure 3.4. With binary classification, if the blue node would predict a different label than the white, this would violate the global monotonicity

constraint since there exists points in the white node that dominate the blue node and vice versa. This problem will not be fixed if we subdivide the white node into multiple leaves by subsequent splits. Thus, since these types of split result in the constraint violation that propagates to child leaves, we decide to avoid them.

It is interesting to note that even if all splits are generated by the non-negative least squares method, it still does not guarantee global monotonicity, but we do, however, have local monotonicity (provided that the dataset is monotone) as can be seen in figure 3.5.

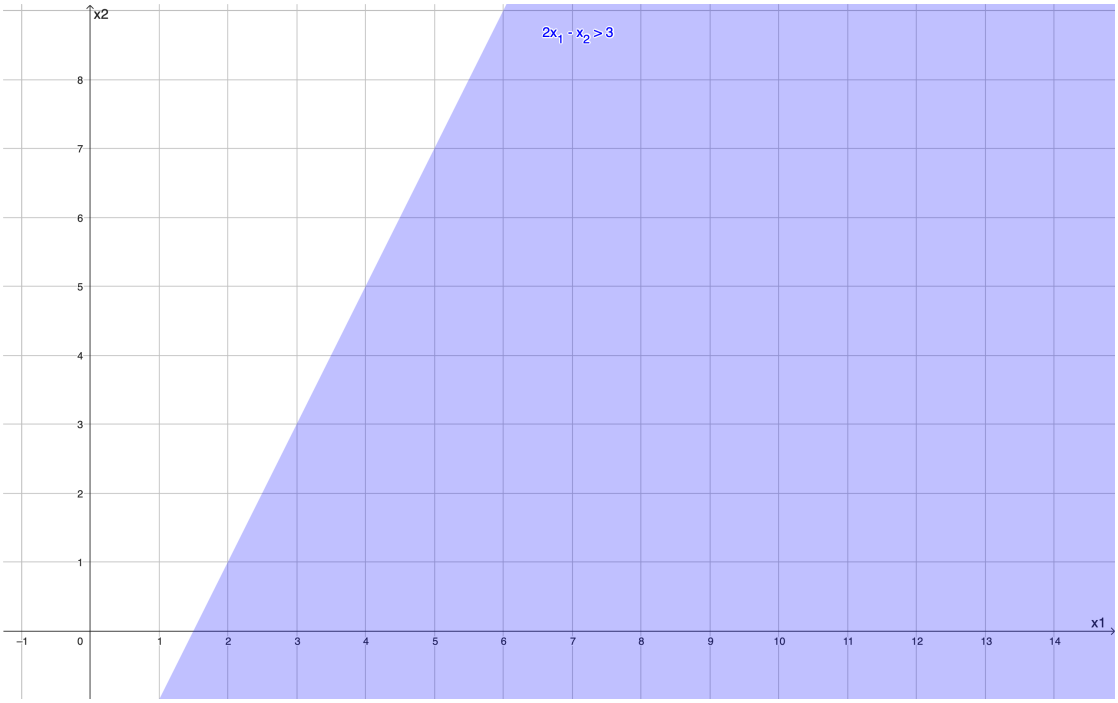


Figure 3.4: Example of a forbidden split. Blue area are all points $\mathbf{x} = (x_1, x_2)$ such that $2x_1 - x_2 > 3$.

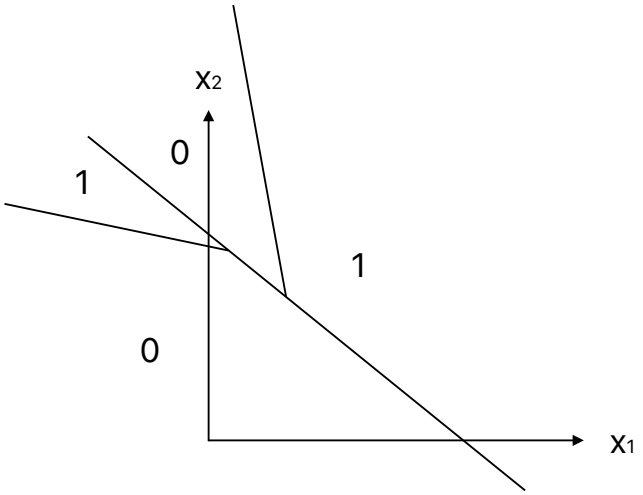


Figure 3.5: Example of using only allowed splits that does not guarantee global monotonicity.

3.4 Partially monotone decision tree using rank mutual information

Two of the authors of [11], Pei and Hu also proposed a partially monotone decision tree based on rank mutual information (RMI) in [12]. They use (slightly modified) RMI scores to determine which features are monotonic, and also to create monotone splits for these features based on RMI. It is in our view better if a domain expert determines which features should be monotone with respect to the target attribute, their algorithm does this exclusively in an automated way based on the given dataset (which may or may not represent the real world).

3.5 Oblique decision tree induction by the cross-entropy optimisation

Bollwein et al. [8] recently discussed a technique that adapts the cross-entropy optimisation method to find near-optimal oblique splits. This is done via a geometrical approach by observing that equivalent oblique splits (splits that lead to the same outcome) can be interpreted as connected regions on a unit hyper-

sphere which is defined by the points in the training data. In each iteration, their algorithm samples multiple candidate solutions from this hypersphere using the von Mises–Fisher distribution which is parameterised by a mean direction and a concentration parameter. These parameters are then updated based on the best-performing samples such that when the algorithm terminates a high probability mass is assigned to a region of near-optimal solutions. Their results show that the algorithm works well for the induction of both compact and accurate oblique decision trees in a small amount of time.

3.6 Convex polytope trees

Armandpour et al. [13] proposed convex polytope trees (CPT) as a generalisation to the class of oblique trees that improves their accuracy and shrinks their size, which consequently provides better predictions. It does so by creating decision boundaries (splitting conditions) based on noisy-OR [14] of multiple linear classifiers (expert-voting model). Just like with an oblique tree, the decision boundaries of a CPT tree are geometrically a convex polytope. They did not attempt to guarantee any local or global monotonicity of the model.

3.7 Induction system for oblique decision trees

Murthy et al. [15] constructed an algorithm called OC1 which combines deterministic hill climbing with two forms of randomisation to find a good oblique split in a given node (top-down approach as usual). The randomisation is there to avoid the hill climber getting stuck in local minima. Simply stated, the algorithm chooses, at a given node, random hyperplanes in the feature space (starting with the best axis-parallel split) and picks the one with the lowest impurity (given some impurity measure). The algorithm performs remarkably well compared to other techniques (such as mentioned in [10]) and it also optimises the impurity reduction, but this method is considerably slower compared to linear regression for oblique splits as discussed in 3.3.

Chapter 4

Methods

In this chapter, our approach to enforcing global monotonicity, as well as the datasets of use are discussed in detail. The algorithm that we construct is called Isotonic Oblique Decision Tree, abbreviated as *IDT-oblique*.

4.1 Setup

Our setting will be a single oblique tree where features could be either categorical or numerical (binary, integer or real numbers). These features can further have a customised domain. For example, house prices are not negative. The target is either a binary class label (0 or 1) in the case of classification or a (continuous) numeric value in the case of regression. In the case of classification, *leaf predictions* are numbers in the interval $[0, 1]$ calculated as the relative frequency of class label 1 of data points of a train set that fall into the specific leaf. If a leaf prediction is greater than or equal to $\frac{1}{2}$, the predicted class label for a data point falling into the leaf is going to be 1 and 0 if it does not fall into the leaf. In other words, the *leaf label* is rounding the leaf prediction to the nearest integer. In the case of regression, the leaf prediction is the mean of the target variable of training observations in a leaf.

Leaves are disjoint by construction. To enforce global monotonicity in our setting, we would like to check whether pairs of leaf nodes violate the monotonicity constraint. Note that a leaf is a conjunction of linear constraints that together form a convex polytope in $\mathcal{X}$. The convex polytopes are disjoint and non-degenerate by construction. The constraint violation happens if for two leaves t, t' we have that $t \preceq t'$, but t' has a higher predicted value (e.g. leaf prediction). If $t \sim t'$, then all pairs of points in the respective leaves are incomparable. The monotonicity constraint is then vacuously satisfied. Simply checking every pair of points inside the respective leaves is infeasible as there could be infinitely many such points.

Constraint programming To determine whether there exists a monotonicity violation between two leaf nodes, we cannot simply look at the minimum and maximum element of the leaves as in [1] since they might not exist in the case of an oblique tree. For example, if the visualisation of a leaf node is an equilateral triangle.

Since a leaf node corresponds to a convex polytope, it can be represented by a set of linear inequalities of the form: $w_1x_1 + w_2x_2 + \cdots + w_px_p \leq c$ or $w_1x_1 + w_2x_2 + \cdots + w_px_p > c$, in p dimensions. Here $\mathbf{w}$ is a vector of coefficients. All the points $\mathbf{x}$ that satisfy these inequalities belong to the leaf node. Our leaf domination problem is formally defined as follows. Let t and t' denote arbitrary leaf nodes of a tree, and let m and n denote the length of the path from the root node to t and t' respectively. Furthermore, let C_{t_i} denote the condition at depth i on the path from the root node to t, where $C(\mathbf{x}) \in \{\mathbf{w}^\top\mathbf{x} \leq c, \mathbf{w}^\top\mathbf{x} > c\}$.

To determine whether t is dominated by t', we verify whether the following set of linear constraints is satisfiable:

(1) $C_{t_1}(\mathbf{x}) \wedge \ldots \wedge C_{t_m}(\mathbf{x})$, and

(2) $C_{t'_1}(\mathbf{x}') \wedge \ldots \wedge C_{t'_n}(\mathbf{x}')$, and

(3) $\mathbf{x} \preceq \mathbf{x}'$.

This problem belongs to the family of *Constraint Satisfaction Problems* [16]. To solve such a problem, we can use a constraint satisfaction system where we want to know whether there exists a feasible solution (points $\mathbf{x}, \mathbf{x}'$) that satisfies the constraints. In case the domain of a feature is integers, we have a mixed integer constraint programming problem. These problems are known to be $\mathcal{NP}$-hard[1]. If all feature-domains are real values, a constraint satisfaction problem belongs to complexity class $\mathcal{P}$. After all, the constraint programming problem can be cast to a linear programming problem by adding a constant as the objective function, and linear programming problems belong to $\mathcal{P}$ [17].

The constraint satisfaction system we use is an API written in Python, which is based on a high-performance theorem prover called Z3[2] [18]. This solver can handle strict inequalities as compared to for example LP-solver of Google OR-Tools[3]. According to the creators of the Z3-solver, it is known as a satisfiability modulo theories (SMT) solver, which works as a sort of extension to SAT solver. One way to think of it is that Z3 contains a SAT solver at its core, augmented by multiple libraries (the 'modulo theories' part of SMT) that allow for formulas which are more complicated than simple literals $(x, \neg x)$ such as $x > 2$ [19]. In other words, the language of SMT solvers is first-order logic.

[1] `http://myweb.usf.edu/~molla//2015_spring_math482/satisfiability.pdf`

[2] `https://ericpony.github.io/z3py-tutorial/guide-examples.htm`.

[3] `https://developers.google.com/optimization/lp/lp_example`.

Oblique tree generation To grow the oblique tree, we use the non-negative least squares method, applied to the data points in each splitting node, as described in section 3.3. The output of this method is (for p numerical features) of the form: $b+a_1x_2+a_2x_2+\cdots+a_px_p$ where the b is a bias term and a_i are coefficients greater than 0 (if a coefficient is 0, the feature is dropped). We will neglect the bias term (effectively setting it to 0) since we use standard split methods [9] for numerical features to determine the best split value. In that case, $a_1x_2 + a_2x_2 + \cdots + a_px_p$ is viewed as being one numerical feature. In the case of classification, we keep splitting until we reach leaf purity, i.e., all leaf nodes contain only data points with the same class label. In the case of regression, splitting to leaf purity could make the tree too large, since the target attribute could assume many different values, and so each leaf will probably contain close to one data point. Already with 150 data points the tree would have somewhere about 150 leaves. For this reason, two variables `nmin` and `minleaf` are introduced to prevent the tree from growing towards leaf purity. If a node contains less than `nmin` data points, then it becomes a leaf node. Furthermore, a split is not allowed if it produces a child node with less than `minleaf` data points.

One might wonder why linear regression is used to compute linear combination splits instead of linear discriminant analysis. The reason is that they are equivalent in the case of binary class labels, save for a bias term which we disregard anyway [20].

Isotonic regression We use the isotonic regression as described in [1] by solving at most m maximal flow problems if the tree contains m leaves. The weights of the leaves are the number of data points (from the train set) that they contain. Isotonic regression is then applied to correct predictions of leaf nodes that violate the monotonicity constraint. We perform isotonic regression on the leaves of a tree only if there are monotonicity violations. The isotonic regression algorithm expects an ordering of the leaves. This ordering is determined via constraint programming as described in the previous paragraph. Then, the algorithm produces for each leaf a new prediction such that it minimises the weighted sum of squared errors with respect to the current leaf predictions, subject to the global monotonicity constraint. This is done for both classification and regression trees. The high-level outline of isotonic regression is found in algorithm 1. Let $\tilde{T}$ denote the set of leaf nodes of tree T, $\bar{y}_t$ denote the average y (target) value of all observations that fall into node t, and $n(t)$ is the number of observations that fall into node t, and D be our leaf order relation.

A new partial ordering is determined as follows:

1. For each pair $t, t' \in \tilde{T}$, determine whether tDt', $t'Dt$, both, or neither.
2. Take the transitive closure of D.

3. Partition $\tilde{T}$ into blocks B such that such that for each $t, t' \in B$: tDt' and $t'Dt$, and B cannot be extended without losing this property. Give each block B the weight and value:

$$w(B) = \sum_{t \in B} n(t), \qquad g(B) = \frac{\sum_{t \in B} n(t)\bar{y}_t}{w(B)}.$$

4. Define the following ordering on the blocks:

$$B_1 \preceq B_2 \Leftrightarrow \exists t_1 \in B_1, t_2 \in B_2 : t_1 D t_2.$$

The relation $(\preceq, \mathcal{B})$ is a partial order. Then, we perform isotonic regression on $(\preceq, \mathcal{B})$ as our order.

Algorithm 1 High-level isotonic regression, given a classification/regression tree T

1: compute the order $(\preceq, \mathcal{B})$ on leaf nodes of T
2: let for each $t \in T$, $g(t)$ be the prediction (which equals mean target value), let $w(t)$ be the number of data points in t
3: compute isotonic regression of function g on order $(\preceq, \mathcal{B})$ with weight function w, which returns function g^*
4: let for each leaf $t \in T$, $g^*(t)$ be the new prediction

Cost complexity pruning We use cost complexity pruning as described in [9] to find the globally monotone oblique tree that has the lowest error rate, using cross-validation.

Four modes of the algorithm The algorithm can be run in four different modes:

1. Oblique tree without any monotonicity constraints,
2. Oblique tree using only non-negative least squares (positive coefficients since we drop features with zero-coefficients) for its linear combination splits, which we refer to as local monotonicity,
3. Oblique tree using only the ICT algorithm to repair predictions of leaf nodes such that we enforce global monotonicity,
4. Oblique tree enforcing both, local and global monotonicity.

The downside of the local monotonicity constraint is that it cannot guarantee a model where the predicted target values always respect global monotonicity, as it is a heuristic. Furthermore, if the dataset is not monotone, then there is no guarantee that the resulting tree satisfies local monotonicity constraint (even though IDT-oblique enforces it). The downside of global monotonicity is that it might be the case that only relatively few data points in a leaf get the 'wrong' label, but isotonic regression changes the label for all points in a leaf and thus it might hurt the performance (e.g. accuracy) of the model.

4.2 Description of IDT-oblique

In this section, the details of IDT-oblique are discussed, following a top-down approach. Relevant parts are discussed and listed in more depth. First, the tree generation is performed as listed in algorithm 2. `X` is the training dataset without labels and `y` is the corresponding labelling.

Algorithm 2 Recursively grow tree starting at the root and creating possible children

1: **if** all instances have the same class **or** all data points are the same **or** node contains less than `nmin` data points **then**
2: current node becomes leaf
3: **return**
4: **end if**
5: **if** oblique_tree == **true then**
6: oblique_feature $\leftarrow$ `linear_regression`$(X, y,$ `non_negative_least_squares`$)$
7: **end if**
8: (split_feature, split_value) $\leftarrow$ `best_split()`
9: determine $X_{left}, X_{right}, y_{left}, y_{right}$
10: recursively create and append left child node and right child node to current node

We further note two important functions: `linear_regression` and `best_split`. The former determines via linear regression an optimal linear combination of features with respect to mean squared error, given whether we want to apply non-negative least squares or not (Boolean value). This linear combination is then applied to the training data to form the new linear combination feature with its values. This feature is considered for splits, thus allowing for possible oblique splits. We further set the intercept equal to 0, following the recommendation of [11] to not include the bias. This is no problem as the calculation to determine the best split introduces its own 'bias'. For example, suppose that the linear regression

procedure gives $3x_1 + x_2$ for some features x_1, x_2. In the case of classification, the decision boundary becomes $3x_1 + x_2 = \frac{1}{2}$. If for a points $\mathbf{x}$: $3x_1 + x_2 \geq \frac{1}{2}$ the predicted label is 1, and 0 otherwise. Observe that this is the same as $3x_1 + x_2 - \frac{1}{2} = 0$ Here, '$\frac{1}{2}$' is the bias term. To determine a split, we only use the left-hand side. Suppose the split is $3x_1 + x_2 - 3 \leq 0$. What essentially happened was that our split calculation method adjusted the bias to maximise impurity (e.g. Gini impurity) reduction. However, if we started with a different bias than $\frac{1}{2}$ the split calculation method would still give $3x_1 + x_2 - 3 \leq 0$. Note further that we essentially have created a new feature $z = 3x_1 + x_2$. Thus we calculate z for each data point, and then z is considered as a 'single' feature for the best split calculation.

Algorithm 3 shows the functions used to determine the best split given the features and their values.

Algorithm 3 Determine best split in a given node

```
(best_feature, best_feature_value) ← None
for feature in features of X do
    if feature is categorical then
        feature_value ← subset of values to split on that maximises impurity reduction
    else
        sort unique values of the feature and check halfway between each pair of consecutive values whether it maximises impurity reduction; assign optimal halfway to feature_value
    end if
    if feature gives a higher impurity reduction than best_feature and left and right child nodes will each have at least minleaf data points then
        (best_feature, best_feature_value) ← (feature, feature_value)
    end if
end for
return (best_feature, best_feature_value)
```

We maximise impurity reduction in algorithm 3. In the case of classification, the Gini impurity (essentially the variance of a Bernoulli random variable) is used. In the case of regression, however, this is the residual sum of squares as explained in chapter 8 of [21].

Thus far, we have explained the basis of IDT-oblique without global monotonicity. To enforce global monotonicity in hindsight, we need to determine the ordering between the leaves of a tree. The ordering is stored in a matrix consisting of zeroes and ones and it is based on our leaf order discussed in chapter 2. The check whether there is a dominance relation between two leaves is done by the

constraint solver as mentioned in the previous section. Let l_1, l_2 be two leaves. We check whether a point that falls into l_2 (respects the splitting rules) dominates (is greater than or equal to) a point that falls into l_1 (in all its coordinates). There is only a slight note to be made if we are in a situation of partial monotonicity (that is, we allow for categorical features). A categorical feature does not necessarily have an ordering between its values. See also subsection 2.2.2. For each categorical feature that is present in the spilling rules of l_1 and l_2, the values should be overlapping for a dominance relation. Should it be that for a categorical feature x, the intersection of the set of values of x found in the splitting rules of l_1 with the set of values of x found in the splitting rules of l_2 is empty, then l_1 and l_2 are incomparable (no dominance relation). If a categorical feature is not present in the splitting rules, we assume that the data points can take on any value of that categorical feature. Consider for example the following situation. Suppose we want to predict house prices in Utrecht with only two features: lot size (continuous number starting from 0) and whether the seller is male or female. Suppose we only have two data points: one where the lot size equals 55 where the seller is male, but the other has a lot size 40 with a female seller. The intersection of the categorical feature is empty and thus we cannot say whether one house is better (dominates) the other.

Given the order matrix between the leaves, the isotonic algorithm described in [1] is used to then correct the predictions of the leaves such that the global monotonicity constraint is enforced. To highlight the important parts, see algorithm 4. The time complexity of this algorithm is asymptotically equal to the time it takes to solve the maximum flow problem on line 19. Given that $|V| \approx |E|$, it does not matter for example whether we use the algorithm of Edmonds-Karp or the push-relabel maximum flow algorithm. We have chosen for Edmonds-Karp. In general, if a graph contains more edges than nodes then push-relabel is more efficient whereas if the graph contains more nodes than edges it is more efficient to use Edmonds-Karp [22]. In our context, if there exist relatively many leaf dominations then push-relabel would be a better choice. However, for simplicity, we have chosen to use Edmonds-Karp regardless of the number of leaf dominations present. Line 19 of the algorithm denotes the return value of the isotonic regression, the new leaf predictions.

To give an example of algorithm 4, consider 4 leaves $V = \{a, b, c, d\}$ and let a be dominated by b, c and d. Let b and c both be dominated by d. Let the function g be defined as

Algorithm 4 Determine new leaf predictions given leaf order matrix X and current leaf predictions; the algorithm is also called the isotonic regression algorithm

construct a graph $G = (V, E)$ where V are the leaves and let E be a directed edge (l_i, l_j) if a point in leaf l_i is dominated by a point in leaf l_j
let $g(x)$ be the current leaf prediction of some leaf $x \in V$
let $w(x)$ be a weight assigned to some leaf $x \in V$
compute $\bar{x} = \sum_{x \in V} \frac{g(x)w(x)}{\sum_{x \in V} w(x)}$
let node capacities be as follows: $c(x) = w(x)(g(x) - \bar{x})$ for $x \in V$
for $x, y \in V$ **do**
    if $x \preceq y$ **then**
        add directed edge with capacity ∞ from x to y
    end if
end for
add two nodes s, t to V
for x **in** $V \setminus \{s, t\}$ **do**
    if $c(x) > 0$ **then**
        add directed edge from s to x with edge capacity $c(x)$
    else if $c(x) < 0$ **then**
        add directed edge from x to t with edge capacity $-c(x)$
    end if
end for
solve maximum flow problem on graph G and find a minimum $S - T$ cut
split graph G in two parts: $L = T \setminus \{t\}$ and $U = S \setminus \{s\}$
if $U \neq \emptyset$ and $L \neq \emptyset$ **then**
    recursively solve on U and L individually
else
    return new leaf predictions $f(x) = \bar{x}$ for $x \in V$
end if

$$g(x) = \begin{cases} \frac{2}{5} & \text{if } x = a, \\ \frac{1}{5} & \text{if } x = b, \\ \frac{4}{5} & \text{if } x = c, \\ \frac{2}{5} & \text{if } x = d. \end{cases} \tag{4.1}$$

Note that these leaves violate the global monotonicity constraint as the leaf prediction of a is $\frac{2}{5}$ which is greater than the leaf prediction of b, which is $\frac{1}{5}$. In addition, the leaf prediction of c is $\frac{4}{5}$ but the leaf prediction of d is smaller, namely $\frac{2}{5}$. Thus leaf b dominates leaf a but has a smaller leaf prediction. Likewise, leaf d dominates leaf c but has a smaller leaf prediction. As for the leaf labelling, the labels assigned to a, b and d are 0 and the label assigned to c is 1. Therefore, in terms of labels, leaves c and d violate the global monotonicity constraint. Suppose further that all weights are set to 1. We have that $\overline{x} = \frac{2}{5}$. After adding the source, sink and edge capacities we end up constructing the directed graph 4.1.

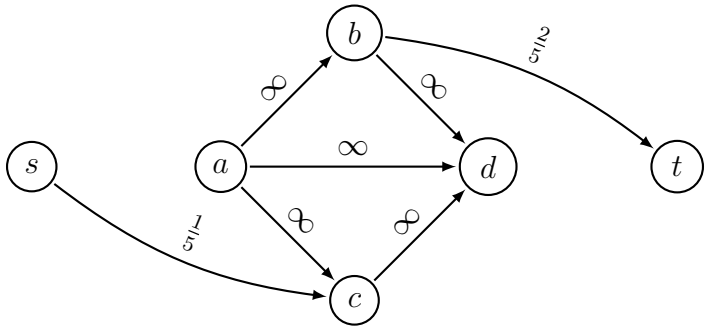


Figure 4.1: Example a of directed graph with 4 leaves, a source and a sink, along with edge capacities.

It is not possible to send flow in the graph from s to t as there is no path from s to t, the max flow is thus equal to 0 and we find the minimum $S - T$ cut to be as follows: $S = \{s, c, d\}$ and $T = \{t, a, b\}$. Thus $L = \{a, b\}$ and $U = \{c, d\}$. As neither sets are empty, we recursively solve the algorithm on the graph projected on a, b and the graph projected on c, d. Starting with the former, we construct graph 4.2. Since a, b are the only leaves we have $\overline{x} = \frac{3}{10}$.

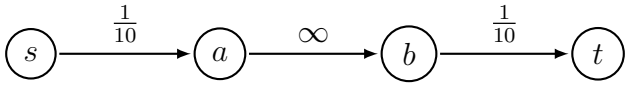


Figure 4.2: Directed graph with edge capacities generated from graph 4.1 using only leaves a, b and adding a source and sink.

We can see that in graph 4.2 the max flow is $\frac{1}{10}$ and we have $S = \{s\}$ and $T = \{a, b, t\}$. As $U = \emptyset$ we set the new predictions of leaves a, b equal to $\frac{3}{10}$.

If we also were to construct a directed graph projected on leaves c and d (and adding a source and sink) we get $\frac{1}{2}$ as new leaf predictions for these two leaves.

Thus we have a new prediction function

$$f(x) = \begin{cases} \frac{3}{10} & \text{if } x = a, \\ \frac{3}{10} & \text{if } x = b, \\ \frac{1}{2} & \text{if } x = c, \\ \frac{1}{2} & \text{if } x = d. \end{cases} \tag{4.2}$$

We see that with the new prediction function, the violations of the global monotonicity constraint have been resolved. As for the leaf labelling, the labels of a and b stay 0, and the label of c stays 1 but the label of d is changed from 0 to 1. With this relabelling, both the predictions and labels respect the global monotonicity constraint. In the end, we are interested in the leaf labels, these should respect the global monotonicity constraint. This happens automatically if we force the leaf predictions to respect the constraint with this algorithm.

As mentioned in the previous section, cost complexity pruning is performed as discussed in [9].

An interesting finding pointed out by [1] is that in the case of classification, when performing the procedure of algorithm 4 two leaves with a common parent might get new predictions such that they end up having the same label. That is, both get a new prediction that is greater than or equal to $\frac{1}{2}$, or both get a new prediction that is less than $\frac{1}{2}$. In that case, we prune back to their common parent, which thus becomes a leaf. However, pruning back to the parent might cause a violation of the global monotonicity constraint. Therefore, we again apply algorithm 4. The procedure of pruning back to the parent and applying ICT is alternated until there are no two leaves with a common parent that have the same prediction (label). This procedure is referred to as *ICT-prune*. See algorithm 5.

Algorithm 6 shows how we use cross-validation in combination with cost complexity pruning, isotonic regression and ICT-prune. We choose to make the tree globally monotone only at the end of the algorithm for the sake of time savings.

There are different ways of doing the cross-validation, for example, we could make every tree in the cost complexity pruning sequence globally monotone before determining the error of β_j. But for the sake of time savings we do isotonic regression only at the end.

Algorithm 5 The ICT-prune procedure

```
change_state ← false
for leaf pair with the same label and common parent p do
    prune back so that p becomes the new leaf
    change_state ← true
end for
if change_state == true then
    apply algorithm 4
    recursively apply ICT-prune again
end if
```

Algorithm 6 The cross-validation procedure

grow a classification/regression tree T_1 on the entire training data
perform cost complexity pruning on T_1 to obtain tree sequence $S = T_1 > T_2 > \cdots > \{t_1\}$ where $\{t_1\}$ denotes the root node and $T_j > T_k$ means that T_k is obtained in by pruning T_j in one or more nodes
for tree T_j in cost complexity tree sequence **do**
    let $\beta_j = \sqrt{\alpha_j \alpha_{j+1}}$ where α_j is a cost parameter obtained via cost complexity pruning
end for
divide training data into 5 equal-sized folds
let F be the set that contains all $\binom{5}{4}$ ways of choosing 4 of the 5 folds
for 4 folds $f \in F$ **do**
    grow a tree T' on f
    perform cost complexity pruning on T' to obtain new tree sequence S'
    for each β_j **do**
        retrieve T'_j from sequence S'
        predict with T'_j on the remaining unused fold, keep track of the errors made for this specific β_j
    end for
end for
choose the β_i that has lowest total error
retrieve T_i from the original cost complexity pruning sequence S
if global monotonicity **then**
    make T_i globally monotone with isotonic regression algorithm 4
    if T_i is a classification tree **then**
        perform ICT-prune algorithm 5
    end if
end if

return T_i

Chapter 5

Experimentation

Our IDT-oblique algorithm discussed in the previous chapter is applied to multiple different datasets. The specific datasets and results are discussed in this chapter.

5.1 Datasets

We use both real and artificially constructed datasets.

5.1.1 Real datasets

Some of the used real datasets are mentioned in section 5.1 of [1]. Within the real datasets, a distinction is made between datasets used for classification and regression. The classification is strictly binary (0/1) and regression is on a numeric target variable. See table 5.1 for the full list of real datasets, along with some statistics. Real datasets are observations of the real world, except for 'Water'. We still include it in the list of real datasets because it describes a scenario which may very well happen in the real world. These datasets are chosen because it is reasonable to think that monotonicity constraints would hold (i.e., the datasets are chosen in such a way that a subset of the features has a common sense monotone relation with the target attribute). As we are not necessarily domain experts, we employ an additional verification step by examining the sign of coefficients obtained through linear regression on a dataset where we predict the specific target value. If the signs of the coefficients comply with common sense, the dataset is included. Using linear regression is better than for example looking at the individual correlation between each feature and the target, as explained in [23]. Linear regression provides an isolated look (all else equal) at the relationship between a feature and the target (even though this may not be a linear relationship in reality, we are only interested in the sign of the coefficient). However, it is not possible to

get an isolated look at the relationship between a feature and the target in terms of correlation. Consider for example two features: the number ice cream sales and the temperature. Let the target be the number of violent crimes. If we look at the individual correlation between the number of ice cream sales and the number of violent crimes, we may get that they are positively correlated. This is, however, not necessarily the case since the temperature could be positively correlated with both features. Thus the correlation between the number of ice cream sales and the number of violent crimes also includes the effects of temperature.

Furthermore, not all features are included. We only selected a subset of the features for which a monotonicity constraint could be justified based on common sense or domain knowledge, and the presumed direction of monotonicity was confirmed by the sign of the linear regression coefficient. Rows with missing values have been removed. Because IDT-oblique makes the harmless assumption that all monotone features have an increasing relationship with the target, if the actual relation is decreasing, the feature values have to be inverted. In case of a monotone-decreasing relation between some feature and the target, we transform the values x as follows

$$x^{`} = x_{max} - x + x_{min} \tag{5.1}$$

where $x^{`}$ is the transformed value and x_{max}, x_{min} are the maximum and minimum values of the feature. The listed datasets are in table 5.1. This includes the cardinality after removing data points with missing values, and the number of features per dataset listed is including the filtering of the features via common sense reasoning and linear regression. The tables in appendix B show for each dataset all features (before feature filtering, and including the target attribute) and their corresponding description.

Real datasets				
Name	Cardinality	Number of used features	Target mean	Target-variance
Bankrupt [24]	440	5	0.5	0.25
Compas [25]	7214	4	0.45065	0.24756
Credit [26]	592	5	0.5	0.25
Haberman [27]	306	3	0.26471	0.19464
Water [28]	1824	9	0.5	0.25
Admission [29]	400	5	0.72	0.02
AutoMPG [30]	392	7	23.45	60.92
Computer [31]	6259	9	2219.58	337333.23
Kuiper [32]	804	5	21343.14	97710314.9
Wages [33]	526	7	5.9	13.64
Windsor [34]	546	11	68121.6	713032634.57

Table 5.1: Real datasets after preprocessing along with statistics. The initial five datasets are for the purpose of classification, whereas the subsequent six datasets are for the purpose of regression.

The Credit dataset is skewed in that only 296 out of 1319 are labelled 0. For this reason, a random sample of size 296 is drawn from the data points with class label 1. Thus the total dataset that we use contains 592 data points. Some features such as 'owner' are encoded as 'no'/'yes' in the original dataset. We modify such values from 'no'/'yes' to 0/1. The same preprocessing is performed with the Bankrupt dataset, it is skewed in that only 220 out of the 6819 are labelled bankrupt. For this reason, a random sample of size 220 is drawn from the remaining non-bankrupt data points. Thus the total dataset that we use contains 440 data points. Dataset Water is skewed in that only 912 out of the 7996 are labelled 1. For this reason, a random sample of size 912 is drawn from the data points with class label 0. Thus the dataset size is reduced to 1824 data points. Finally, with the Compas dataset, we have changed the value 'Male' of the 'sex' attribute to 1 and 'Female' to 0. For 'c_charge_degree', the 'M' is changed to 0 and 'F' to 1. Even though the Haberman dataset is skewed distributed (relative frequency of class 1 of 0.26), We decide to not take a sample from it as the cardinality is already relatively small.

We now give two detailed examples of what we mean by common sense monotone relation with the target attribute and use of linear regression, using the Bankrupt and Credit datasets mentioned in table 5.1.

Bankrupt The target attribute of this dataset is to determine whether companies are bankrupt or not (1 if yes, 0 if not). All the features are normalised in the

range 0 to 1 by changing the values per future using the formula in equation 5.2 [35].

$$x^{`} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{5.2}$$

Here, x is the old feature value, $x^{`}$ is the transformed value and x_{max}, x_{min} are the maximum and minimum values of the feature in discussion.

Using common sense reasoning with linear regression as validation, the following features are chosen: 'Debt ratio %', 'Current Liability to Current Assets', 'Cash Flow to Liability', 'Net Income to Total Assets' and 'Fixed Assets to Assets'. See also table 5.2 for the outcome of linear regression. We rely on the book "Corporate Finance" by Berk and DeMarzo (2011) to explain these ratios [36].

'Debt ratio %' is defined as $\frac{\text{Liability}}{\text{Total Assets}}$. Firstly, as the debt ratio increases, it indicates that a larger proportion of a company's assets are financed by liability (such as loans) rather than equity. This implies a higher risk of bankruptcy. Secondly, a higher debt ratio means that a company has a larger amount of debt obligations to fulfil. If the company faces financial difficulties or economic downturns, it may struggle to meet its debt obligations, increasing the likelihood of bankruptcy. Additionally, a high debt ratio can affect a company's creditworthiness and ability to obtain additional financing. Lenders may view a high debt ratio as a signal of financial instability. This further enhances the risk of bankruptcy.

Regarding 'Current Liability to Current Assets', the ratio indicates the proportion of a company's short-term liabilities compared to its short-term assets. A higher ratio suggests difficulty for a company to cover immediate financial obligations, increasing its risk of bankruptcy.

Cash flow refers to the total amount of cash that flow in minus out of a company during a specific period. It is important to look at the cash flow because accounting rules might allow a company to recognise revenue before services or sales are executed [37]. These recorded events have not generated cash yet. Thus, to get an overview of the financial resources available to a company, it is important to look at the cash flow. A higher 'Cash Flow to Liability' suggests that the company has enough cash flow to maintain its operations and repay debts. In contrast, a lower ratio may indicate financial instability and potential difficulties in covering its liabilities, increasing the likelihood of bankruptcy.

The ratio '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. In contrast, a lower ratio indicates lower profitability, potentially signalling financial distress (income no longer fulfils financial obligations) which on its turn can lead to an increased risk of bankruptcy.

Finally, 'Fixed Assets to Assets' indicates the proportion of a company's assets

that are allocated to fixed or long-term assets, such as buildings and machinery. A higher ratio implies that a larger portion of a company's assets are invested in long-term assets. If a company faces financial difficulties or economic downturns, it may struggle to meet its short-term debt obligations, because fixed assets cannot easily be converted to cash, increasing the likelihood of bankruptcy.

Both 'Cash Flow to Liability' and 'Net Income to Total Assets' have a monotone decreasing relation with the target, so the corresponding feature values are modified using equation 5.1 to get a monotone increasing relation.

Feature	Coefficient
Debt ratio %	3.18
Current Liability to Current Assets	1.26
Cash Flow to Liability	-2.01
Net Income to Total Assets	-1.77
Fixed Assets to Assets	ϵ

Table 5.2: Linear regression on the Water dataset. Here, ϵ stands for a very small positive number.

Credit dataset The dataset contains originally 11 features and 1 target attribute. The target is simply whether the application for a credit card is accepted given the feature values (1 if accepted, 0 if not). The dataset contains 1319 instances without missing values.

It stands to reason that the banks tend to accept low-risk applicants [38]. So, if there are negative reports (major derogatory reports) about the applicant, the bank is less likely to accept the application. Thus we expect to see a minus sign for the coefficient of the feature 'reports'. For 'age', it is not that obvious. On the one hand, if the applicant is relatively younger then he is more likely to be able to work and cover the credit card expenses. But on the other hand, older people tend to be more responsible for their expenses. For this reason, 'age' is left out. Of course, if the income is relatively high, the bank would likely see no reason to assume that the applicant will abuse the credit card later on. The 'share' feature is the ratio of monthly credit card expenditure to yearly income. One would reason that if this ratio is relatively high, a bank would be more inclined to accept the credit card application because the credit card would apparently play an important role in someone's expenditure and likely be used frequently (or used in high-price scenarios). Furthermore, using the credit card frequently implies a higher per-transaction profit for the bank (that stems from additional fees for using the credit card). The feature 'homeowner' stands for whether an applicant owns a home. This could indicate a stable financial situation of the applicant and is a better risk for the bank as it has more grip on the applicant, so we expect to

see a positive coefficient sign. Feature 'dependents' stands for how many people depend on the applicant's income. This could indicate that the applicant has a stable financial situation, but the applicant may also be unmarried and/or have no children. This feature can lead to different interpretations and is therefore left out. If an applicant is self-employed, then there is no necessary guarantee for a source of income. Thus we expect a minus coefficient sign for 'selfemp'. The feature 'months' indicate how many months an applicant lives at their current address. We reason that the longer an applicant lives at their current address, the more likely he is going to stay there and is thus a good ('safe') risk for banks. This is however not a compelling argument in our opinion, thus this feature is left out. The feature 'majorcards' stands for the number of major credit cards the applicant currently holds. If the applicant already has multiple credit cards, then this implies that other banks find the applicant to be a good risk. We do not find this a compelling argument, however, thus the feature 'majorcards' is left out. The feature 'expenditure' stands for the mean monthly credit card expenditure. We see no compelling reason as to why this has a monotone relation with the target. Finally, 'active' stands for the number of active credit accounts of an applicant, which is also left out. Indeed, when we observe the linear regression output in table 5.3 we get that each coefficient sign fits our common sense.

Feature	Coefficient
reports	-0.08
income	0.02
share	2.47
owner	0.15
selfemp	-0.12

Table 5.3: Linear regression on the Credit dataset.

Note that for IDT-oblique we want to have monotone increasing relations as mentioned above in this subsection, thus we change the feature values accordingly with equation 5.1. To sum up, all the features in table 5.3 are used in our experiments.

Following the same line of reasoning as in the paragraph above, we discuss the other datasets as follows.

Compas For this dataset, the target attribute is the expected recidivism (new arrest within two years, after release) of a criminal (1 if we expect it to happen, 0 if not). The features chosen are 'age', 'sex', 'priors_count' and 'c_charge_degree'. Because 'age' has a decreasing monotone relation, its feature values are modified so that the relationship becomes monotone increasing.

Haberman The target attribute is the survival status of patients within 5 years (1 if the patient survived, 2 if the patient died). Because the dataset contains 1/2 labels instead of 0/1, we modified it to 0/1. All features are used for this dataset.

Water The target attribute is whether the water in an urban environment is safe or not (1 if safe, 0 if not). With common sense, we expect all features to have a monotone decreasing relation with the target since all features are dangerous toxins. Yet, with linear regression, not all coefficient signs are negative. See table 5.5. Some coefficients are positive. The reason for these unexpected positive coefficients could be because we took a sample from the entire dataset (due to the skewness), noise in the data, the model may not be linear at all, or strong correlation between (independent) features that cause this effect, also called multicollinearity. However, the sign of the coefficients when performing linear regression is the same if we consider the entire dataset before preprocessing, see table 5.4. However, after also considering figure 5.1 (which is created on the Water dataset after preprocessing) multicollinearity seems plausible. See for example the strong correlation of 0.72 between 'viruses' and 'bacteria'.

After experimenting and removing some features we decided to choose all features listed in table 5.6. Since all coefficients are negative, we changed the feature values accordingly with equation 5.1.

Feature	Coefficient
aluminium	0.09
ammonia	$-\epsilon$
arsenic	-0.23
barium	0.01
cadmium	-1.31
chloramine	0.02
chromium	0.13
copper	-0.03
flouride	0.01
bacteria	0.07
viruses	-0.09
lead	-0.13
nitrates	$-\epsilon$
nitrites	-0.02
mercury	-3.04
perchlorate	$-\epsilon$
radium	$-\epsilon$
selenium	-0.37
silver	-0.09
uranium	-0.88

Table 5.4: Linear regression on the Water dataset, before any preprocessing. Here, ϵ stands for a very small positive number.

Feature	Coefficient
aluminium	0.11
ammonia	$-\epsilon$
arsenic	-0.43
barium	0.02
cadmium	-3.07
chloramine	0.04
chromium	0.29
copper	-0.08
flouride	0.01
bacteria	0.13
viruses	-0.24
lead	-0.45
nitrates	-0.01
nitrites	-0.08
mercury	-2.22
perchlorate	$-\epsilon$
radium	-0.01
selenium	-1.17
silver	-0.37
uranium	-2.15

Table 5.5: Linear regression on the Water dataset. Here, ϵ stands for a very small positive number.

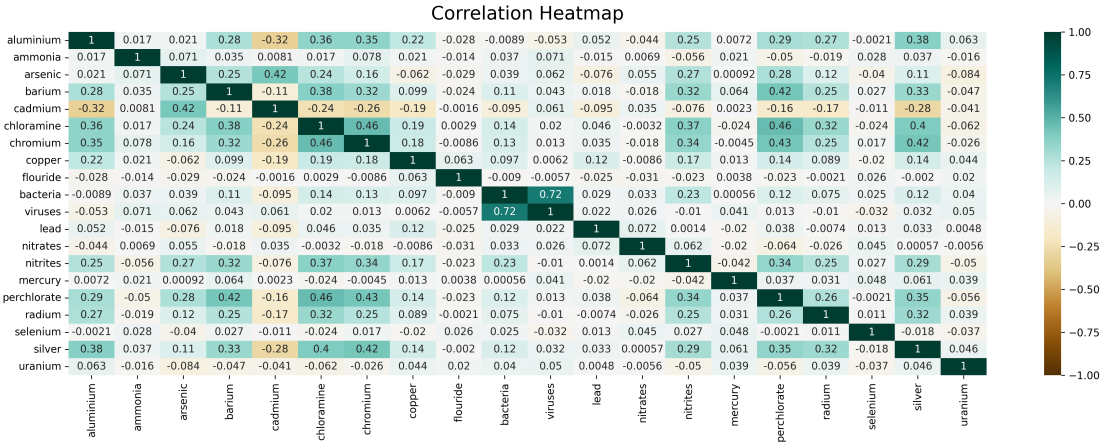


Figure 5.1: Correlation matrix of the features of the Water dataset.

Feature	Coefficient
ammonia	$-\epsilon$
arsenic	-0.12
cadmium	-6.03
copper	-0.01
viruses	-0.16
lead	-0.35
nitrates	-0.01
mercury	-3.9
selenium	-1.04
uranium	-2.1

Table 5.6: Linear regression on the Water dataset using only a subset of the features. Here, ϵ stands for a very small positive number.

Admission The target attribute is a number between 0 and 1 which represents the probability to be admitted to some university. The 'University rating' feature should have a negative monotone relation with the target: the higher the rating, the lower the probability to get admitted. All other features should be chosen with common sense, and they all should have a monotone increasing relation with the target because according to this dataset, the higher the scores and ratings, the better the performance of the student. However, this turned out not to be the case as can be seen in table 5.7. The 'University rating' has a positive coefficient sign, therefore we remove it. We then redo the linear regression without 'University rating' and we observe that the 'SOP' still has a negative coefficient. As 'SOP' stands for 'statement of purpose' we find its negative coefficient contradicting our common sense (we expect to see a positive coefficient). Thus, we decided to not include 'SOP' in our experiments. The selected features are found in table 5.8.

Feature	Coefficient
GRE Score	ϵ
TOEFL Score	ϵ
University Rating	0.01
SOP	$-\epsilon$
LOR	0.02
CGPA	0.12
Research	0.02

Table 5.7: Linear regression on the Admission dataset using only a subset of the features. Here, ϵ stands for a very small positive number.

Feature	Coefficient
GRE Score	ϵ
TOEFL Score	ϵ
LOR	0.02
CGPA	0.12
Research	0.02

Table 5.8: Linear regression on the Admission dataset using only a subset of the features as in table 5.7, but without 'University Rating' and 'SOP'.

AutoMPG The target attribute is the miles per galon (MPG) of a car, which is an integer number. The 'car name' and 'origin' are not included for they are nominal features. It should not matter what the origin of the car is when we are only interested in monotonicity. Similarly, there are a lot of unique names thus this would imply that IDT-oblique cannot say anything about the relation between two cars just because their name is different. Observe table 5.9 The 'displacement' feature should have a monotone decreasing relation with the target, yet the coefficient using linear regression is positive. However, as an exception, we still keep this feature because the reason it has a negative coefficient is due to a strong correlation with the other features, see table 5.10 [39]. The 'acceleration' is the rate at which a car can increase its speed, common sense tells us that the coefficient should be negative. However, according to the table, it is positive, so we decided to drop that feature from our experiments. Thus all features minus 'acceleration' in the table are included in our experiments.

Feature	Coefficient
cylinders	-0.33
displacement	0.01
horsepower	$-\epsilon$
weight	-0.01
acceleration	0.09
model year	0.75

Table 5.9: Linear regression on the AutoMPG dataset using only a subset of the features. Here, ϵ stands for a very small positive number.

	class	cylinders	displacement	horsepower	weight	acceleration	model year
class	1	−0.78	−0.81	−0.78	−0.83	0.42	0.58
cylinders	−0.78	1	0.95	0.84	0.90	−0.50	−0.35
displacement	−0.81	0.95	1	0.90	0.93	−0.54	−0.37
horsepower	−0.78	0.84	0.90	1	0.86	−0.69	−0.42
weight	−0.83	0.90	0.93	0.86	1	−0.42	−0.31
acceleration	0.42	−0.50	−0.54	−0.69	−0.42	1	0.29
model year	0.58	−0.35	−0.37	−0.42	−0.31	0.29	1

Table 5.10: Correlation matrix for AutoMPG dataset.

Computer The target attribute of this dataset is the price of a personal computer. The dataset contained a mistake however, the values of the 'FIRM' feature are inverted (meaning that instead of 1 they had 0, and instead of 0 they had 1). This was rectified. All of the features are chosen as they have a monotone increasing relation with the target. The 'TREND' feature stands for how long the computer is available in the market. Logically, it has a monotone-decreasing relation with the target, so its values are modified to obtain a monotone-increasing relation with the target.

Kuiper The target attribute of this dataset is the suggested retail price of a car. The chosen features are 'Mileage', 'Cylinder', 'Cruise', 'Leather' and 'Type'. Feature 'Mileage' has a monotone decreasing relation with the target, so its values are modified to obtain a monotone increasing relation with the target. This dataset is included as 'Type' (e.g. sedan or coupe) is a categorical (nominal) feature, making it the only dataset in our collection that has a categorical feature.

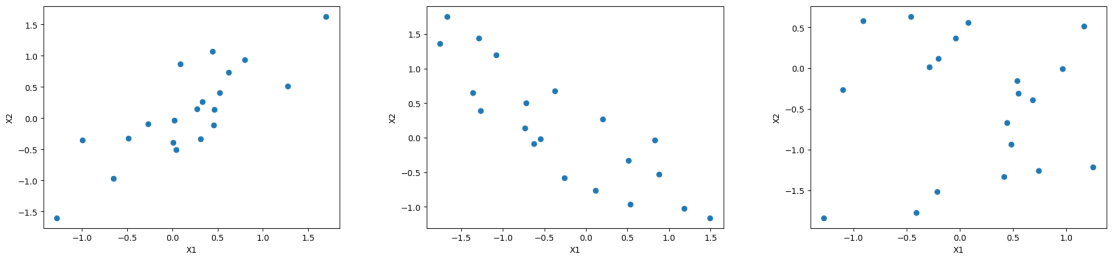
Wages The target attribute of this dataset is the hourly wage of a worker. The features chosen are 'educ', 'exper', 'tenure', 'female', 'nonwhite', 'numdep', and 'married'. Note, however, that there is a decreasing monotone relation of 'female' with the target, and of 'nonwhite' with the target. The features (corresponding 0/1 values) are therefore changed to 'male' and 'white'. This way, all chosen features have an increasing monotone relation with the target.

Windsor The target attribute of this dataset is the price of a house. Some features such as 'aircon' (whether central air conditioning is available) are encoded as 'no'/'yes' in the original dataset. We modify such values from 'no'/'yes' to 0/1. All features should have a monotone increasing relation with the target, and indeed this is confirmed via linear regression as all feature coefficients have a positive sign. Thus, all features of this dataset are included.

5.1.2 Artificial datasets

Artificial datasets are constructed only for classification and used to study the relationship between properties of the data and performance of IDT-oblique. The artificial datasets are constructed in two phases. First, the dataset without labels is constructed, and then random monotone labellings per dataset are produced such that the relative frequency of the majority class is at most 60%.

Three types of datasets are drawn from a multivariate normal distribution, one with a high positive (0.9) correlation between the features (from now referred to as *positive dataset*), one with zero correlation between the features (from now referred to as *zero dataset*) and one with a high negative correlation between the features (from now referred to as *negative dataset*). With the latter, we create two groups of equal size that constitute the feature set, such that the features are highly positively correlated (0.9) within their respective groups, but highly negatively correlated (-0.9) between the groups. This way, the positive dataset is more likely to give us many comparable data points (using our product order), see figure 5.2a. The negative dataset is more likely to give us few comparable points, see figure 5.2b. The zero dataset is to observe the working of IDT-oblique in a random setting, see figure 5.2c. See for example figure The number of features is either 2, 6 or 10. The number of data points is 2000.



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

Figure 5.2: Small illustrative datasets of 20 data points and 2 features each, with different degrees of correlation between the features. Figure 5.2a has 175 comparable pairs of points (dominance relation between two points), figure 5.2b has 49 comparable pairs of points and figure 5.2c has 115 comparable pairs of points.

After a dataset is constructed, multiple random monotone binary labellings are generated. Because this is a random procedure, multiple labellings are selected instead of one. This way, coincidences from drawing random labellings are averaged out as we are interested in the general performance of IDT-oblique. The procedure for generating a single random monotone binary labelling is called *Propp-Wilson* and it is a Markov chain Monte Carlo algorithm, using a "sandwiching" technique

as follows. Let X be a set of data points. A set $L \subseteq X$ is a lower set of X if $\mathbf{x} \in L$ and $\mathbf{x}' \preceq \mathbf{x}$ implies $\mathbf{x}' \in L$. We start two Markov chains: one from the empty lower set and one from X. Then, given some number of iterations, we do the following. In each iteration, we draw a random data point from X, and toss a fair coin to decide whether we will add the data point to both lower sets, or remove the data point to both lower sets. Note that we only perform the add/remove operation to a lower set if the resulting set is still a lower set. We do this until the sets "meet" (the lower sets contain the same data points and are thus the same); if they did not meet within the specified number of iterations, the number of iterations is doubled and the procedure is repeated. The algorithm is proven to converge [40]. There is a one-to-one correspondence between lower sets of X and monotone binary labellings on X: give all elements of the lower set class label 0, and all other elements class label 1. This way, we have generated a random monotone binary labelling. In appendix A, the algorithm is discussed in its generality along with pseudocode. To give a concrete example, consider figure 5.3. In this example we have have $X = \{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}\}$ and lower set $L = \{\mathbf{a}, \mathbf{b}\}$. If the data points in L get class label 0 and the data points in $X \setminus L$ get class label 1, then this binary labelling is monotone.

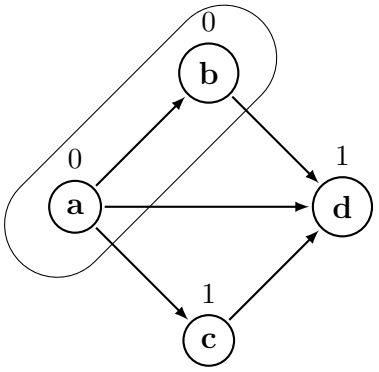


Figure 5.3: Example of a lower set with 2 data points: **a** and **b**. An arrow from **a** to **b** indicates that $\mathbf{a} \preceq \mathbf{b}$. The numbers displayed above the nodes represent the class labels.

After generating random monotone binary labellings, a dataset is split into train/test sets. At most 5% noise is introduced independently to the train/test to make the datasets more realistic. First, we determine the number of pairs of points such that there is a domination relation between points within a pair. These pairs need further to be disjoint, meaning that a data point can be present in at most one pair. Lastly, the labels of the points within a pair need to be the same (either both points are 0 or both are 1). Then, noise is introduced in one pair as follows. Let the points in one pair be called $\mathbf{x}, \mathbf{y}$ and suppose that point $\mathbf{x} \preceq \mathbf{y}$. Then,

if both labels are 0, we change the label of $\mathbf{x}$ to 1. Otherwise, if both labels are 1, we change the label of $\mathbf{y}$ to 0. The amount of noise we introduce divided by the size of the train/test set is at most 5% (meaning that 5% is the goal, but if we could not find that many pairs to introduce noise with, we introduce as much noise as possible). This method to introduce noise is from now on referred to as the *disjoint pairs method*.

This method of noise introduction guarantees that the number of labels that need to be re-labelled to regain monotone labelling is precisely equal to the amount of noise introduced. See the following lemma 5.1.1.

Lemma 5.1.1. *The number of data points that need to be re-labelled to regain the monotonicity of the labelling equals the amount of noise introduced using the disjoint pairs method.*

Proof. First, a pair is only considered for noise if there exists a domination relation between the points of the pair. Observe further that for a given pair where noise is introduced, the monotonicity of the labels cannot be fixed if we change the labels of points outside this pair. The only way to restore monotonicity between the points of the pair is to re-label one point of the pair. Since the pairs are disjoint, the total number of re-labels necessary to regain the monotonicity of the labelling is at least equal to the total amount of noise introduced.

The total number of re-labels necessary to regain the monotonicity of the labelling is of course at most the amount of noise introduced: simply undo the noise.

Thus, the number of labels that need to be re-labelled to regain the monotonicity of the labelling equals the amount of noise introduced using the disjoint pairs method. □

To conclude this subsection, the splitting of a dataset into train/test tests in case of a classification task is done in a stratified way on the labels. This means that the train set and test set will have roughly the same labelling distribution. This is useful since otherwise we might train/test a tree on skew data, which could lead to unfair conclusions on the performance of the tree.

5.1.3 Train set size and regulation

In all of our experiments, three different train set sizes are taken into consideration (determined via preliminary experimentation): 50, 100 or 150. The reason for these three sizes is that if the train set is large enough, then the regression tree tends to satisfy the constraints without explicitly enforcing them. Consequently, it becomes hard to see the effect of enforcing monotonicity constraints via IDT-oblique for larger train sets. If the dataset is 50 then we set `nmin` $= 2$, `minleaf` $= 1$. If the

dataset is 100 then we set `nmin` = 8, `minleaf` = 4. If the dataset is 150 then we set `nmin` = 12, `minleaf` = 6. The values for `nmin` and `minleaf` are determined via preliminary experimentation such that the growth of a tree does not take too long, and we do not have a lot of leaves which helps to speed up IDT-oblique.

5.2 Approach

To reiterate, all four modes of IDT-oblique are applied to the datasets. We perform cost-complexity pruning using cross-validation. For classification, we also perform ICT-pruning as explained earlier in the modes where global monotonicity is enforced.

Statistics For classification tasks, we use the following metrics:

- Accuracy
- Precision
- Recall
- F_1 $\left(\text{which is equal to } 2\frac{precision \cdot recall}{precision + recall}\right)$
- Area under ROC curve (auroc)

For regression tasks, we settle with the mean squared error (MSE) as a metric. Statistical significance testing can be performed in multiple ways. For example, McNemar's tests compare all pairs of the modes of the tree or pairwise t-tests. But there is a downside, as there are so many multiple tests, the probability of a type 1 error also increases. To counter that Bonferroni correction could be applied, which is however considered conservative and weak [41]. Since we have multiple trees (different modes of IDT-oblique), and would like to compare them over multiple datasets, we follow the recommendation of [41] to use the Friedman test to check whether we can reject a null-hypothesis (significance level at $\alpha = 0.05$ as standard done in these tests). In the case we reject it, a post-hoc significance test called Nemenyi is applied to know which mode of IDT-oblique performs significantly better compared to the others. Since the Nemenyi only gives a statistically significant difference between the test statistics (not which mode is better compared to the other), we manually look at the test statistics and see which mode contains higher values compared to another mode. According to the same paper, these two tests work very well together and are one of the best choices for our use case. Another benefit of using these two tests is that in comparisons over multiple datasets, we do not have to deal with an elevated type 1 error, as the sources of

the variance are the differences in performance over (independent) datasets and not on (usually dependent) samples.

The statistic for significance testing in the case of classification is accuracy. The reason for that is that this metric is more intuitive than for example F_1 which combines two other metrics: precision and recall. A higher F_1 score is good as it implies that a model's prediction has relatively few false positives and few false negatives. Accuracy is a simple metric that only denotes how many times a model is correct (proportion of true positives and true negatives with respect to test set size). As the used classification/artificial datasets are balanced with respect to their class labels, our choice fell on accuracy.

There are multiple ways to train a tree and determine its performance statistics as mentioned above. For example, splitting a dataset in train/validation/test set. In this case, we would train a regression/classification tree on the training data, then prune our tree and pick the one with the lowest error on the validation set. Finally, the statistics are determined by applying this tree to the test set. There is however one major drawback of using this conventional approach: the cardinality of the dataset needs to be large enough so that the results are more reliable. Given that our real datasets are sometimes quite small, such as the Haberman dataset which contains relatively few data points, this train/validation/test split might not be the best choice. We could remedy this problem by augmenting the dataset with bootstrap sampling. However, bootstrap sampling does not treat the original dataset as if it is the population as it involves sampling with replacement from the original dataset. Thus, an augmented dataset by bootstrap sampling does not contain more information about the population than what is given in the original dataset. For that reason, it may not work well with small datasets (i.e., we do not gain additional information and thus the results are still potentially unreliable). Instead, for real datasets we opt for implementing the cross-validation technique for regression/classification trees as explained in section 4.2. But for artificially constructed datasets, since we have enough data points, we apply a train/validation/test set split on the dataset.

5.2.1 Train/validation/test set

First, we train a classification/regression tree on the train set. Then a pruning sequence is determined. Then the best tree (based on accuracy) is chosen by applying the trees of the tree sequence to the validation set. In the case of global monotonicity, ICT-prune is applied in the chosen tree. Finally, the required statistics are determined by applying the chosen tree to the test set. The validation set size is set to be 50% of the size of the dataset minus the train set. The test set contains the remaining 50% of the data points.

5.2.2 Cross-validation

With cross-validation we split a dataset as train/test set. To highlight an important part of algorithm 6, for each β_j we follow along the pruning sequence and retrieve the corresponding tree and with it we perform predictions on the remaining fold which was left out. This way, for each β_j, we have 5 out-of-sample predictions (as one fold was left out each time), and we choose the corresponding tree that has the lowest total error. That is, we sum the error of all 5 out-of-sample predictions together and determine which β_j corresponds to the lowest error and pick the corresponding tree (after possible isotonic regression/ICT-prune) from the original/first tree sequence. Its error is estimated as the following ratio: $\frac{\text{total error from cross-validation}}{\text{size of dataset}}$. More details are found in [42]. In literature, this method is considered good and useful [43]. To illustrate the working with a small example (no local or global monotonicity constraint), consider table 2.1 again. Here, we get the following α sequence: $\alpha_1 = 0, \alpha_2 = 0.1, \alpha_3 = 0.4$. Then we can compute the βs as $\beta_1 = \sqrt{0 \cdot 0.05} = 0, \beta_2 = \sqrt{0.1 \cdot 0.4} = 0.2, \beta_3 = \sqrt{0.4 \cdot \infty} = \infty$. By performing cross-validation we get a total error of 3 for both β_1 and β_2. For β_3 we get an error of 9. As both β_1, β_2 are equally good, we can choose either of them, say β_1 (even though β_2 is a smaller tree). The total error of the corresponding tree is estimated as $\frac{3}{10} = 0.3$.

The chosen tree from the train set is then applied to the test set to calculate the desired statistics.

5.3 Results and discussion

Certain values in the classification and artificial result tables are highlighted in grey. In the case of classification, this indicates the mode or modes of the IDT-oblique with the highest accuracy on the test set being considered. In the case of regression, this indicates the mode or modes with the smallest mean squared error. In our evaluation tables, the abbreviation LG refers to the mode where both local and global monotonicity constraints are enforced. $\bar{L}G$ corresponds to the mode where only the global monotonicity constraint is enforced. $L\bar{G}$ represents the mode where only the local monotonicity constraint is enforced. Finally, $\bar{L}\bar{G}$ indicates the fully unconstrained mode. Furthermore, in the tables where the p-value is smaller than the significance level, those values are also highlighted in grey. Additional statistical tables regarding classification and artificial datasets can be found in appendix D.

5.3.1 Real data

For the classification task, the labels of the train and test set are stratified and statistical significance testing is conducted with accuracy as metric. For regression tasks, the target is not stratified (as it is not a label anymore), and statistical significance testing is conducted with MSE as metric. The training size is 150 for each dataset. We have the following results. For statistical significance testing, we take as null hypothesis H_0 = All modes of IDT-oblique perform equally well on the datasets with respect to accuracy in the case of classification; MSE in the case of regression. The significance level is set to $\alpha = 0.05$.

Regression Table 5.11 shows the MSE for each mode of IDT-oblique applied on the regression test sets, for all train set sizes.

We can deduce from these tables that the smaller the train set is, the more prominent the effect of enforcing monotonicity constraints as in general the difference of the MSE values between LG and $\bar{L}\bar{G}$ tends to decline on the test sets. For example, with AutoMPG the difference is 0.85 when train set size is 50. When the train set size is 100, the difference is 0.73. When the train set is 150, the difference is 0.21. This phenomenon has been discussed in subsection 5.1.3: If the train set is large enough, then the regression tree tends to satisfy the constraints without explicitly enforcing them.

Furthermore, for all training set sizes, the mode where both monotonicity constraints are enforced seems to be the top performer as it has the lowest MSE in 4 out of the 6 datasets (for train set size 100 even 5 out of 6). The mode where the only local monotonicity constraint is enforced comes in second place. Only once do the modes that do not enforce the local monotonicity constraint beat the rest, when the train set size is 150 with respect to the Kuiper dataset, $\bar{L}\bar{G}$ is the best performing mode. This seems to be an outlier in our results. A reason could be that the chosen features do not represent the need for enforcing (global) monotonicity constraint enough. Or, if we were to enforce global monotonicity, it affects prediction of the entire leaf. This implies that enforcing the global monotonicity constraint would only correct for a small number of incorrect labels for data points in a leaf while introducing relatively more wrong new labels for the rest of the leaf. Another reason could be that the global monotonicity constraint is more often vacuously satisfied due to difference of data points in the values for the categorical feature 'Type'.

The statistical significance testing results of comparing the IDT-oblique modes are as follows. If the train size is 50, we get a p-value from 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 5.12 for the post-hoc Nemenyi test. We can conclude that there is a statistical significance between

mode $\bar{L}G$ compared to mode LG. If we look at the individual MSE values for both modes in table 5.11, we can conclude that the MSE values are smaller for LG and thus it performs significantly better than the $\bar{L}G$. Similarly, mode $\bar{L}G$ is significantly worse based on MSE than $L\bar{G}$. From this, we deduce that the local monotonicity constraint is more critical than the global monotonicity constraint. Even though there is no significant difference, we still see that the MSE is lower in the case of the mode LG compared to $\bar{L}G$.

Proceeding with significance testing if the train size is 100, we get a p-value from the Friedman test equal to 0.051. This is larger than our α and thus we cannot reject H_0 in this case.

For train test size 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 5.13. 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 [41]. We reject the H_0 but cannot give a conclusive answer which mode of IDT-oblique outperforms another, although one may assume from table 5.13 that LG significantly outperforms $\bar{L}G$ (looking at the respective MSE values gives an indication which mode outperforms the other). So having the local monotonicity constraint (all coefficients are positive in oblique splits) has its benefits if the train set size is relatively large.

We see that if the train set size declines, the differences between the modes of IDT-oblique become more significant, indeed giving more ground to our theory that if the train set is large enough, then the regression tree tends to satisfy the constraints without explicitly enforcing them.

Finally, in general, not enforcing the local monotonicity constraint generates a worse model compared to enforcing it. This is likely due to the tree making splits early on with negative coefficients, which results in an unrecoverable split as discussed in section 3.3. Another reason is that enforcing local monotonicity will likely set the negative coefficients to zero, effectively using less features when determining the splits. This combats overfitting of the model on the data.

	Train set size	LG	$\bar{L}G$	$L\bar{G}$	$\bar{L}\bar{G}$
Admission	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
AutoMPG	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
Computer	50	142200	351550	212403	218503
	100	94112	106919	97988	116586
	150	87466	92163	88271	95291
Kuiper	50	101475000	103377100	64724450	69963970
	100	106261200	85241970	49858290	50385930
	150	103304700	103198100	40536170	38513820
Wages	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
Windsor	50	363860200	420322200	412652000	574990300
	100	311893800	415459900	325382200	413842200
	150	284492000	334268700	299826600	335350300

Table 5.11: Test set MSE for regression datasets.

	LG	$\bar{L}G$	$L\bar{G}$	$\bar{L}\bar{G}$
LG	1			
$\bar{L}G$	0.014	1		
$L\bar{G}$	0.9	0.014	1	
$\bar{L}\bar{G}$	0.23	0.66	0.23	1

Table 5.12: The p-values of the post-hoc Nemenyi test for regression where train set size equal to 50.

	LG	$\bar{L}G$	$L\bar{G}$	$\bar{L}\bar{G}$
LG	1			
$\bar{L}G$	0.07	1		
$L\bar{G}$	0.9	0.11	1	
$\bar{L}\bar{G}$	0.28	0.9	0.4	1

Table 5.13: The p-values of the post-hoc Nemenyi test for regression where train test size equal to 150.

Classification Table 5.14 shows the accuracies for each mode of IDT-oblique applied on the classification test sets, for all train set sizes.

In the classification case we witness the same phenomenon as with regression: the smaller the train set is, the more prominent the effect is of enforcing monotonicity constraints. For example, with Bankrupt the difference between LG and $\bar{L}\bar{G}$ is 0.05 when train set size is 50. When the size of the train set is 100, the difference is 0.01. However, when the train set size increases to 150, the difference becomes 0. It should be noted that for certain datasets like Compas, the difference between 100 and 150 may actually increase. Nevertheless, overall, when considering the trend from 50 to 150, the differences tend to decrease. For train set sizes 50 and 150, the LG mode is the best performer (or ties with other modes) on all datasets. Only if train set size 100, the LG mode has 0.001 less accuracy compared to $\bar{L}\bar{G}$ on Compas. Given the relatively small difference, this could be a coincidence.

Observe further that for train set sizes 50 and 100 the $\bar{L}G$ mode has an accuracy of 0.5 on the test set of Water. This is due to the pruning of the specific tree back to the root. In that case, the prediction is the majority class in the train set. Since we use stratified sampling, the class labels of data points are evenly distributed in the train set, resulting in about 50% chance of predicting the correct label. We would like to point out another interesting observation concerning the Water dataset, for example, if the train set size is 50. When we do not enforce any monotonicity constraint, the split that occurs at the root (which also happens to be the only split in the entire model) can be found in equation 5.3.

$$\begin{aligned} &0.009 \cdot \text{ammonia} - 0.64 \cdot \text{arsenic} + 9.4 \cdot \text{cadmium} - 0.07 \cdot \text{copper} \\ &- 0.23 \cdot \text{viruses} - 1.06 \cdot \text{lead} - 0.017 \cdot \text{nitrates} + 37.18 \cdot \text{mercury} \\ &+ 1.18 \cdot \text{selenium} - 0.59 \cdot \text{uranium} \leq 0.41 \end{aligned} \tag{5.3}$$

We see that monotonicity constraints are necessary because split 5.3 does not comply with common sense: if there are more viruses in the water, the water would be safer according to the model (remember the features have been transformed such

that they are monotone increasing with the target, thus the plus and minus signs need to be read vice versa).

The accuracies for Haberman perhaps do not mean as much since the dataset has a relative frequency of class 1 of 0.26 (see table 5.1).

On Credit all of the modes perform equally well. Consider as an example the following tree that is constructed without enforcing any monotonicity constraints, and without pruning (train set size 50 and tree is traversed via depth-first traversal).

```
Node level) splitting rule to current node ; #data points ;
    impurity ; relative frequency of class 1 ; star (*) if node is
    leaf
0) root ; 50 ; 0.25 ; 0.5
1) share <= 0.0016 ; 26 ; 0.037 ; 0.0385
2) 0.0102*reports-0.0101*income+565.0641*share+0.2496*homeowner
    -0.1033*selfemp <= 0.5861 ; 25 ; 0.0 ; 2.22e-16 ; *
2) 0.0102*reports-0.0101*income+565.0641*share+0.2496*homeowner
    -0.1033*selfemp > 0.5861 ; 1 ; 0.0 ; 1 ; *
1) share > 0.0016 ; 24 ; 0.0 ; 1 ; *
```

The second split of the tree is a complex oblique split that uses all of the features. This split lets the tree overfit on the train set, and no further splitting is necessary afterwards as the child leaves are pure. However, due to cost complexity pruning, we prune back such that the tree consists only of one root and two leaves. In that case, the only feature taken into consideration is 'share'. Thus, the ratio of monthly credit card expenditure to yearly income becomes the deciding factor as to whether a credit card application is accepted. That is why, if train set size is the same for all modes, the accuracy on Credit is the same for all modes. This is because all models generate a tree with only one split on the 'share' feature. More examples of generated trees for various datasets are found in appendix C.

The statistical significance testing results of comparing the IDT-oblique modes are as follows. For train set size 50 the p-value of Friedman test is 0.039, this means that we reject H_0. See table 5.15 for the p-values of the post-hoc Nemenyi test. Although the p-values are all greater than α due to the inexactness of the post-hoc test, it can be deduced that mode LG is significantly better than $\bar{L}\bar{G}$ based on accuracy.

For train set size 100 the p-value is 0.26 and therefore we cannot reject H_0.

For train test size 150 the p-value of the Friedman test is 0.39, again we can not reject H_0.

We can come to the conclusion that the various modes of IDT-oblique exhibit comparable performance in classification tasks when considering the accuracy metric and train set sizes of 100 and 150. However, upon observing the tables, it is evident that enforcing both local and global monotonicity constraints results in an overall better performance for these train set sizes. This improvement could

be attributed to either the absence of decisive features in our experiments, or the datasets lacking a decisive feature altogether. Nevertheless, for a train set size of 50, we discovered a significant improvement by enforcing both monotonicity constraints when compared to the fully unconstrained counterpart.

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

Table 5.14: Test set accuracy for classification datasets.

	LG	$\bar{L}G$	$L\bar{G}$	$\bar{L}\bar{G}$
LG	1			
$\bar{L}G$	0.32	1		
$L\bar{G}$	0.6	0.9	1	
$\bar{L}\bar{G}$	0.07	0.87	0.6	1

Table 5.15: The p-values of the post-hoc Nemenyi test for classification with train set size equal to 50.

5.3.2 Artificial data

In this subsection, the results for the artificially constructed datasets are discussed. The same hypothesis and statistical significance testing from the previous subsection is adopted. Some statistics about the generated labels (before introducing noise) are provided in tables D.13, D.14 and D.15 of appendix D. The datasets are

all for the purpose of classification. The different modes of IDT-oblique are run on each degree of correlation (positive/zero/negative). For each correlation degree, the metric (accuracy) is the mean of the metric taken over all labellings for that particular degree of correlation (and number of features). The results are found in table 5.16. In general, we do not observe the same trend as with the real datasets: a smaller train set does not seem to make enforcing monotonicity constraints more prominent. However, we make the following two observations.

1. All else equal, accuracy tends to decline with the number of features. This effect is most emphatic with respect to the zero and negative datasets. For example, if train set size is 50 and the correlation is negative, then we have for the mode LG the following accuracies: $\{(2, 0.813), (6, 0.581), (10, 0.497)\}$, where the first item of a tuple is the number of features and the second item of the tuple is the corresponding accuracy value.

2. All else equal, accuracy tends to increase with correlation between the features. For example, if the train set size is 150 and the number of features is 2, then we have for the mode LG the following accuracies: $\{(\text{negative}, 0.84), (\text{zero}, 0.9), (\text{positive}, 0.92)\}$.

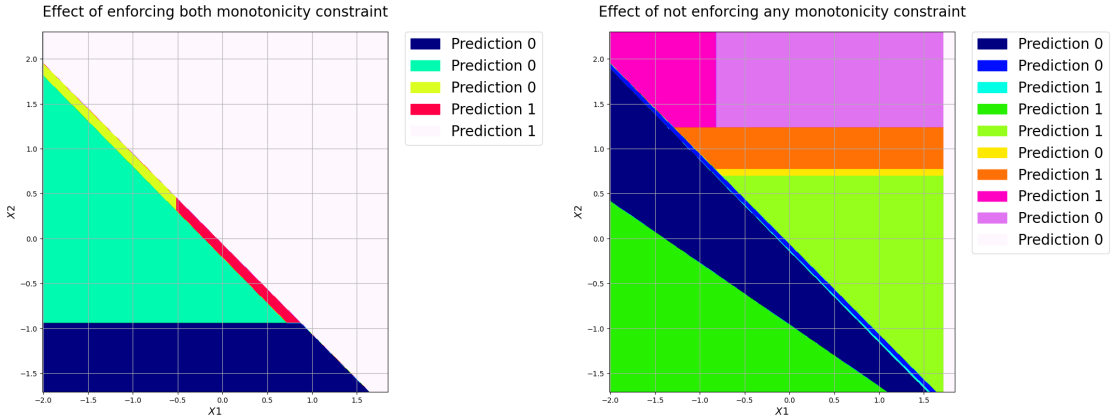
A plausible explanation for the first observation is that for a fixed correlation degree between the features (e.g. positive correlation), having fewer features implies more comparable pairs of data points, as two points $\mathbf{x}, \mathbf{x}'$ are only comparable if $\mathbf{x} \preceq \mathbf{x}'$ or $\mathbf{x}' \preceq \mathbf{x}$. If there are p features, then $\mathbf{x} \preceq \mathbf{x}'$ holds if and only if $x_i \leq x'_i$ for all $1 \leq i \leq p$. For a fixed correlation degree, the data is randomly generated. Thus, if there are relatively many features, then it is more probable to have $x_i > x'_i$ for some i. To validate this observation, we conduct a statistical test between the number of features. For each fixed number of features there are 3 degrees of correlation, 4 modes of IDT-oblique and 3 different test set sizes; constituting a total of 36 accuracy values. We set $\alpha = 0.05$. The p-value of Friedman test is $2.32 \cdot 10^{-16}$. See table 5.17, 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.

As for the second observation, a plausible explanation is that higher correlation implies more comparable pairs, see figure 5.2. To validate this observation, we again conduct a statistical test but now between the correlation degrees. For each fixed degree of correlation, there are 36 accuracy values. Again we set $\alpha = 0.05$. The p-value of Friedman test is $1.04 \cdot 10^{-14}$. See table 5.18, the performance on positive datasets is significantly better than on zero or negative, and the performance on zero datasets is significantly better than on negative.

Also worth noting is that the mode LG performs better on positive/zero datasets as compared to negative datasets. Just like with real datasets, enforcing monotonicity constraints is favourable.

Observe also that in table 5.16 we have an accuracy of 0.494 on the negative dataset where the number of features is 50 and train set size is 50 with mode $L\bar{G}$. An accuracy worse than 0.5 means that this model is worse (accuracy-wise) than just predicting a label randomly in a uniform way. Also unique is that the accuracy value on this negative dataset worsens when both local and global monotonicity constraints are enforced, compared to only enforcing the global monotonicity constraint.

To give an illustration of how enforcing monotonicity constraints can make the tree smaller, see figure 5.4. In the example we have two features, negative correlation between them, a train set size of 50. In one case we enforce both monotonicity constraints (LG), in the other we have the fully unconstrained counterpart ($\bar{L}\bar{G}$). The dataset used is entitled `2features.pickle_negative_labelling_9.csv` and can be found in our supplemented GitHub repository.



(a) Leaves of mode LG.

(b) Leaves of mode $\bar{L}\bar{G}$.

Figure 5.4: Illustrated leaves of classification trees, generated with two modes of IDT-oblique: LG and $\bar{L}\bar{G}$. The train set size used is 50 and there are two features with negative correlation between them. No cost complexity pruning is performed.

Statistical significance testing between the modes of IDT-oblique is conducted as follows. The same hypothesis as classification is adopted and all modes of IDT-oblique are tested with respect to all datasets given a particular train set size. Per train set size, there are 9 datasets since for each correlation degree we have a variant with 2,6 or 10 features. Thus per mode, we have a list of 9 accuracies. The results are as follows.

If the train set size is 50, then the p-value of the Friedman test is 0.03. The post-hoc Nemenyi test however could not find significant differences, see table 5.19, but it is safe to assume that it is between LG and $\bar{L}G$. The Friedman test probably found a significant difference between these modes and a manual observation of

table 5.16 tells us that LG performs significantly better based on accuracy than $\bar{L}G$.

If the train set size is 100, then the p-value of the Friedman test is 0.31. Thus all modes perform equally well, which makes sense when we inspect the accuracy tables.

If the train set size is 150, then the p-value of the Friedman test is 0.0053. See table 5.20 for the p-values of the post-hoc Nemenyi test. Mode $L\bar{G}$ is significantly better compared to $\bar{L}G$. Thus only enforcing the local monotonicity constraint is significantly better for accuracy as opposed to only enforcing the global monotonicity constraint.

	Number of features	Train set size	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 5.16: Test set accuracy for artificial datasets.

	2	6	10
2	1		
6	0.001	1	
10	0.001	0.001	1

Table 5.17: The p-values of the post-hoc Nemenyi test between the number of features.

	Positive	Zero	Negative
Positive	1		
Zero	0.001	1	
Negative	0.001	0.001	1

Table 5.18: The p-values of the post-hoc Nemenyi test between the correlation degrees.

	LG	$\bar{L}G$	$L\bar{G}$	$\bar{L}\bar{G}$
LG	1			
$\bar{L}G$	0.052	1		
$L\bar{G}$	0.9	0.13	1	
$\bar{L}\bar{G}$	0.26	0.88	0.46	1

Table 5.19: The p-values of the post-hoc Nemenyi test based on accuracy where train set size is 50.

	LG	$\bar{L}G$	$L\bar{G}$	$\bar{L}\bar{G}$
LG	1			
$\bar{L}G$	0.082	1		
$L\bar{G}$	0.9	0.018	1	
$\bar{L}\bar{G}$	0.46	0.77	0.18	1

Table 5.20: The p-values of the post-hoc Nemenyi test based on accuracy where train set size is 150.

Chapter 6

Conclusion

The objective of this research has been to develop the first monotone oblique decision tree algorithm, known as IDT-oblique. This was achieved by incorporating isotonic regression and constraint solving. Isotonic regression is applied to the order matrix of the leaves based on a domination relation between them, and the leaf predictions, ensuring that the new leaf predictions uphold a monotone decision tree. The dominance relation is determined by recognising that the leaves can be represented as convex polytopes in the feature space, allowing them to be expressed as linear inequalities. These inequalities are then given to the constraint solver, Z3, along with an inequality expressing whether a point in leaf t dominates (is greater than) a point in leaf t' based on the product order.

In addition to guaranteeing monotonicity, the option to only allow oblique splits with positive coefficients has also been incorporated. This is because, without this constraint, a split could occur in a way that cannot be corrected for in subsequent splits, potentially requiring a relabelling of a significant portion of the leaves (see section 3.3).

IDT-oblique has been evaluated by conducting experiments on well-known datasets for monotone classification and regression, as well as artificially constructed data.

6.1 Answers to research questions

In the first chapter, we formulated three research questions that we will address in this section. Research question 1 inquired about how we can verify whether a given oblique tree is monotone. This is achieved by constructing the leaf order matrix using the Z3 constraint solver. Subsequently, for each pair of leaves, we check whether the leaf prediction correspond with the dominance relation stored in the matrix. If leaf t dominates leaf t', the mean target value in the case of

regression (leaf label in the case of classification) of t should be at least as high as that of t'.

The second research question pertained to generating a decision tree in a justified manner. Justification comes from not randomly assigning new monotone predictions, but rather optimising for performance by minimising the weighted sum of squared errors with respect to the current leaf predictions, subject to the global monotonicity constraint.

The final research question examined the predictive performance of our tree with enforced monotonicity constraints compared to the unconstrained counterpart. We have measured this using the MSE metric with respect to regression and the accuracy metric with respect to classification.

In the case of regression, we observed that when IDT-oblique enforces both local and global monotonicity constraints, it significantly outperforms the other modes that do not enforce them, in most cases. Furthermore, the smaller the train set is, the more prominent the effect is of enforcing monotonicity constraints, as the difference in MSE of the algorithm mode that enforces both monotonicity constraints compared to its fully unconstrained counterpart tends to decline on the test sets. If the train set is large enough, then a regression tree tends to satisfy the constraints without explicitly enforcing them. The Kuiper dataset seems to be an outlier to this. A reason could be that the chosen features do not represent the need for enforcing monotonicity constraint enough. Another reason could be that the global monotonicity constraint is more often vacuously satisfied due to difference of data points in the values for the categorical feature. Furthermore, for a relatively small train set size of 50, the mode that enforces local monotonicity but not global monotonicity significantly outperforms the mode that does not enforce local monotonicity but enforces global monotonicity. In general, not enforcing the local monotonicity constraint generates a worse model compared to enforcing it. This is likely due to the tree making splits early on with negative coefficients, which results in an unrecoverable split. Another reason is that enforcing local monotonicity will likely set the negative coefficients to zero, effectively using less features when determining the splits. This combats overfitting of the model on the data.

In the case of classification we observe the same phenomenon as with regression: a smaller train set makes enforcing monotonicity constraints more prominent and favourable in terms of accuracy. For train set size of 50 we find a significant improvement by enforcing both monotonicity constraints compared to the fully unconstrained counterpart. For larger train set sizes, the gain in accuracy is very minimal. The unconstrained model tends to implicitly adhere to the monotonicity present in the training data. However, it might also be due to the fact that we are not domain experts in the fields where the data is collected, and we might have

missed crucial features. Additionally, we have seen that monotonicity constraints are necessary for the explanation of the model. For the Water dataset, not enforcing the local monotonicity constraints leads to a model that contradicts common sense: if there are more viruses in the water, the water would be safer according to an unconstrained model. In general, the mode where both monotonicity constraints are enforced produces the best results compared to the other modes.

In the case of artificial datasets (also classification tasks), we have made two observations. We discovered that all else equal, accuracy tends to significantly decline with the number of features. This effect is most emphatic with respect to the zero and negative datasets. The second observation is that, all else equal, accuracy tends to significantly increase with correlation between the features. A plausible explanation for both of these observations is that higher correlation/fewer number of features implies more comparable pairs of data points. Furthermore, the accuracy on the negatively correlated dataset with 10 features and a train set size of 50 is worse than random guessing of a class label (accuracy of 0.494). Additionally, accuracy for this negatively correlated dataset worsens when both local and global monotonicity constraints are enforced, compared to only enforcing global monotonicity. We have not encountered a situation with real data where enforcing both constraints is worse compared to only enforcing the global monotonicity constraint.

We have also observed that, in general, enforcing the monotonicity constraints improves the performance of IDT-oblique compared to not enforcing them, assuming the underlying nature of the data adheres to monotone relationships between the features and the target.

6.2 Future work

The IDT-oblique algorithm focuses on trees for classification tasks with binary labelling. A potential extension could involve incorporating any ordinal labelling by exploring the First-order Stochastic Dominance discussed in [1], or the Kotlowski approach discussed in [2].

Another straightforward extension is enabling an ensemble of trees, such as a random forest, to reduce bias, as ensemble models tend to be more complex [44].

Further research could also investigate the generated unconstrained oblique splits. These allow for negative feature-coefficients. Currently, no form of regularisation is implemented. However, it is apparent that an unconstrained oblique split always uses all features, which can lead to overly complex splits that overfit the training data. Consequently, this can result in IDT-oblique performing worse, as seen with artificial datasets when there is a relatively large number of features. One possible solution might involve incorporating regularisation and tuning the

regularisation parameter correctly (e.g., through grid search), allowing only a subset of the features to be used.

Finally, an open question is whether it is possible to assume that all numeric feature domains are real valued for constraint solving, so that the problem will belong to $\mathcal{P}$.

Bibliography

[1] R. van de Kamp, A. Feelders, and N. Barile, "Isotonic classification trees," in *Advances in Intelligent Data Analysis VIII, 8th International Symposium on Intelligent Data Analysis, IDA 2009, Lyon, France, August 31 - September 2, 2009. Proceedings*, N. M. Adams, C. Robardet, A. Siebes, and J. Boulicaut, Eds., ser. Lecture Notes in Computer Science, vol. 5772, Springer, 2009, pp. 405–416. DOI: 10.1007/978-3-642-03915-7_35. [Online]. Available: https://doi.org/10.1007/978-3-642-03915-7%5C_35.

[2] C. Bartley, W. Liu, and M. Reynolds, "Enhanced random forest algorithms for partially monotone ordinal classification," in *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, AAAI Press, 2019, pp. 3224–3231. DOI: 10.1609/aaai.v33i01.33013224. [Online]. Available: https://doi.org/10.1609/aaai.v33i01.33013224.

[3] W. Duivesteijn and A. Feelders, "Nearest neighbour classification with monotonicity constraints," in *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML/PKDD 2008, Antwerp, Belgium, September 15-19, 2008, Proceedings, Part I*, W. Daelemans, B. Goethals, and K. Morik, Eds., ser. Lecture Notes in Computer Science, vol. 5211, Springer, 2008, pp. 301–316. DOI: 10.1007/978-3-540-87479-9_38. [Online]. Available: https://doi.org/10.1007/978-3-540-87479-9%5C_38.

[4] J. R. Cano, P. A. Gutiérrez, B. Krawczyk, M. Wozniak, and S. Garcia, "Monotonic classification: An overview on algorithms, performance measures and data sets," *CoRR*, vol. abs/1811.07155, 2018. arXiv: 1811.07155. [Online]. Available: http://arxiv.org/abs/1811.07155.

[5] R. Potharst and A. J. Feelders, "Classification trees for problems with monotonicity constraints," *SIGKDD Explor.*, vol. 4, no. 1, pp. 1–10, 2002. DOI: 10.1145/568574.568577. [Online]. Available: https://doi.org/10.1145/568574.568577.

[6] M. J. Pazzani, S. Mani, and W. R. Shankle, "Acceptance of rules generated by machine learning among medical experts," *Methods of information in medicine*, vol. 40, no. 05, pp. 380–385, 2001.

[7] "Decision trees." (2013), [Online]. Available: https://blackquest.files.wordpress.com/2013/12/lecture24.pdf (visited on 06/23/2022).

[8] F. Bollwein and S. Westphal, "Oblique decision tree induction by cross-entropy optimization based on the von mises-fisher distribution," *Comput. Stat.*, vol. 37, no. 5, pp. 2203–2229, 2022. DOI: 10.1007/s00180-022-01195-7. [Online]. Available: https://doi.org/10.1007/s00180-022-01195-7.

[9] A. Feelders. "Classification trees." (2021), [Online]. Available: http://www.cs.uu.nl/docs/vakken/mdm/trees-2021.pdf (visited on 12/07/2022).

[10] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and regression trees*. Routledge, 2017.

[11] S. Pei, Q. Hu, and C. Chen, "Multivariate decision trees with monotonicity constraints," *Knowl. Based Syst.*, vol. 112, pp. 14–25, 2016. DOI: 10.1016/j.knosys.2016.08.023. [Online]. Available: https://doi.org/10.1016/j.knosys.2016.08.023.

[12] S. Pei and Q. Hu, "Partially monotonic decision trees," *Inf. Sci.*, vol. 424, pp. 104–117, 2018. DOI: 10.1016/j.ins.2017.10.006. [Online]. Available: https://doi.org/10.1016/j.ins.2017.10.006.

[13] M. Armandpour, A. Sadeghian, and M. Zhou, "Convex polytope trees and its application to VAE," in *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021, pp. 5038–5051. [Online]. Available: https://proceedings.neurips.cc/paper/2021/hash/285a25c17f351708754cdb6d56f3962e-Abstract.html.

[14] J. Pearl, *Probabilistic reasoning in intelligent systems*. Elsevier, 2014, vol. 88.

[15] S. K. Murthy, S. Kasif, and S. Salzberg, "A system for induction of oblique decision trees," *J. Artif. Intell. Res.*, vol. 2, pp. 1–32, 1994. DOI: 10.1613/jair.63. [Online]. Available: https://doi.org/10.1613/jair.63.

[16] C. Lecoutre, *Constraint Networks: Targeting Simplicity for Techniques and Algorithms*. John Wiley & Sons, 2013.

[17] N. Karmarkar, "A new polynomial-time algorithm for linear programming," *Comb.*, vol. 4, no. 4, pp. 373–396, 1984. DOI: 10.1007/BF02579150. [Online]. Available: https://doi.org/10.1007/BF02579150.

[18] B. Dutertre and L. M. de Moura, "A fast linear-arithmetic solver for DPLL(T)," in *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, T. Ball and R. B. Jones, Eds., ser. Lecture Notes in Computer Science, vol. 4144, Springer, 2006, pp. 81–94. DOI: 10.1007/11817963_11. [Online]. Available: https://doi.org/10.1007/11817963%5C_11.

[19] N. Bjørner and L. de Moura. "The inner magic behind the z3 theorem prover." (2019), [Online]. Available: https://www.microsoft.com/en-us/research/blog/the-inner-magic-behind-the-z3-theorem-prover/ (visited on 05/06/2023).

[20] G. S. Maddala, *Limited-dependent and qualitative variables in econometrics*. Cambridge university press, 1983.

[21] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 112.

[22] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009, ISBN: 978-0-262-03384-8. [Online]. Available: http://mitpress.mit.edu/books/introduction-algorithms.

[23] M. Magdon-Ismail and J. Sill, "A linear fit gets the correct monotonicity directions," *Mach. Learn.*, vol. 70, no. 1, pp. 21–43, 2008. DOI: 10.1007/s10994-007-5028-4. [Online]. Available: https://doi.org/10.1007/s10994-007-5028-4.

[24] *Bankrupt dataset*, https://www.kaggle.com/datasets/fedesoriano/company-bankruptcy-prediction, Accessed: 2023-05-05.

[25] *Compas dataset*, https://www.kaggle.com/datasets/danofer/compass, Accessed: 2023-05-05.

[26] *Credit dataset*, https://www.kaggle.com/datasets/dansbecker/aer-credit-card-data, Accessed: 2023-05-05.

[27] *Haberman dataset*, https://www.kaggle.com/datasets/gilsousa/habermans-survival-data-set, Accessed: 2023-05-05.

[28] *Water dataset*, https://www.kaggle.com/datasets/mssmartypants/water-quality, Accessed: 2023-05-05.

[29] *Admission dataset*, https://www.kaggle.com/datasets/akshaydattatraykhare/data-for-admission-in-the-university, Accessed: 2023-05-05.

[30] *Autompg dataset*, https://www.kaggle.com/datasets/uciml/autompg-dataset, Accessed: 2023-05-05.

[31] *Computer dataset*, http://qed.econ.queensu.ca/jae/2006-v21.3/stengos-zacharias/, Accessed: 2023-05-05.

[32] *Kuiper dataset*, https://jse.amstat.org/jse_data_archive.htm, Accessed: 2023-06-09.

[33] *Wages dataset*, https://rdrr.io/cran/wooldridge/man/wage1.html, Accessed: 2023-05-05.

[34] *Windsor dataset*, https://www.kaggle.com/datasets/photosho/house-prices-for-the-city-of-windsor-canada, Accessed: 2023-05-05.

[35] D. Liang, C. Lu, C. Tsai, and G. Shih, "Financial ratios and corporate governance indicators in bankruptcy prediction: A comprehensive study," *Eur. J. Oper. Res.*, vol. 252, no. 2, pp. 561–572, 2016. DOI: 10.1016/j.ejor.2016.01.012. [Online]. Available: https://doi.org/10.1016/j.ejor.2016.01.012.

[36] J. Berk and P. DeMarzo, *Corporate Finance: Global Edition*. Pearson Education, 2011.

[37] M. C. Inc. "Accounting and auditing requirements for taiwan company or branch." (2013), [Online]. Available: http://www.matchest.com/en_US/aboutusDetail.asp?id=20#:~:text=Businesses%5C%20are%5C%20required%5C%20to%5C%20maintain,(IFRS)%5C%20and%5C%20US%5C%20GAAP..

[38] W. H. Greene, "A statistical model for credit scoring," 1992.

[39] A. Feelders. "Data mining introduction." (2022), [Online]. Available: http://www.cs.uu.nl/docs/vakken/mdm/Slides/dm-intro2022.pdf.

[40] J. G. Propp and D. B. Wilson, "Exact sampling with coupled markov chains and applications to statistical mechanics," *Random Structures & Algorithms*, vol. 9, no. 1-2, pp. 223–252, 1996.

[41] J. Demsar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, 2006. [Online]. Available: http://jmlr.org/papers/v7/demsar06a.html.

[42] A. Feelders. "Classification trees (2)." (2021), [Online]. Available: http://www.cs.uu.nl/docs/vakken/mdm/Slides/dm-classtrees-2-2021.pdf (visited on 05/06/2023).

[43] H. Blockeel and J. Struyf, "Efficient algorithms for decision tree cross-validation," *J. Mach. Learn. Res.*, vol. 3, pp. 621–650, 2002. [Online]. Available: http://jmlr.org/papers/v3/blockeel02a.html.

[44] A. Feelders. "Bagging and random forests." (2021), [Online]. Available: http://www.cs.uu.nl/docs/vakken/mdm/Slides/dm-bias-variance2021.pdf (visited on 06/17/2023).

Appendices

Appendix A

Algorithm for generating monotone labelling

Let $(X, \preceq)$ be a partially ordered set. A set $L \subseteq X$ is a lower set of X if $x \in L$ and $x' \preceq x$ implies $x' \in L$. Let $\mathcal{L}$ denote the collection of all lower sets of X. Note that there is a one-to-one correspondence between lower sets of X and monotone binary labellings on X: give all elements of the lower set the label 0, and all other elements the label 1. The downset of x is defined as $\downarrow x = \{x' \in X : x' \preceq x\}$. Likewise, the upset of x is $\uparrow x = \{x' \in X : x \preceq x'\}$. If L is a lower set, then $L \cup x$ is a lower set if and only if $\downarrow x \subseteq L$. Likewise, if L is a lower set, then $L \setminus x$ is a lower set if and only if $\uparrow x \cap L = \emptyset$.

See algorithm 7 for the pseudocode. The algorithm performs a random walk on a state space, where each state corresponds to a lower set of X. We start two Markov chains: one from the empty lower set (all elements of X get label 1), and one from X (all elements of X get the label 0). These chains are coupled in the sense that they use the same random numbers. In each iteration we draw a random element from X, and toss a fair coin to decide whether we will add or remove this element. If the two chains "meet" (end up in the same state) in (less than) T iterations, we have a random draw from $\mathcal{L}$. Otherwise, we double the number of iterations, and run the chains again from their initial states. It is important that the same random numbers are used in the same iteration numbers of different runs of the chains. In the pseudocode this is done by running the chain from time T down to time 1, and storing the random draws in the vectors $\mathbf{x}$ and $\mathbf{u}$.

Algorithm 7 Generate random lower set

$T \leftarrow 1$
draw $\mathbf{x}[1]$ uniformly at random from X
draw $\mathbf{u}[1]$ uniformly at random from $\{0, 1\}$
repeat
  $L_1 \leftarrow \emptyset$
  $L_2 \leftarrow X$
  for $t = T$ **downto** 1 **do**
    $x \leftarrow \mathbf{x}[t]$
    $u \leftarrow \mathbf{u}[t]$
    if $u = 1$ **then**
      if $L_1 \cup x \in \mathcal{L}$ **then**
        $L_1 \leftarrow L_1 \cup x$
      end if
      if $L_2 \cup x \in \mathcal{L}$ **then**
        $L_2 \leftarrow L_2 \cup x$
      end if
    else
      if $L_1 \setminus x \in \mathcal{L}$ **then**
        $L_1 \leftarrow L_1 \setminus x$
      end if
      if $L_2 \setminus x \in \mathcal{L}$ **then**
        $L_2 \leftarrow L_2 \setminus x$
      end if
    end if
  end for
  $T \leftarrow 2 \times T$
  $\mathbf{x} \leftarrow$ append($\mathbf{x}$, sample(X, size $= T/2$))
  $\mathbf{u} \leftarrow$ append($\mathbf{u}$, sample($\{0, 1\}$, size $= T/2$))
until $L_1 = L_2$
return L_1

Appendix B

Tables of all datasets with their features and description

These tables are derived from the datasets of table 5.1. The features include the target attribute in these tables. Note that the features included here are prior to feature filtering.

Bankrupt	
Feature	Description
target	Bankrupt?: Class label 0/1
X1	ROA(C) before interest and depreciation before interest: Return On Total Assets(C)
X2	ROA(A) before interest and percentage after tax: Return On Total Assets(A)
X3	ROA(B) before interest and depreciation after tax: Return On Total Assets(B)
X4	Operating Gross Margin: Gross Profit/Net Sales
X5	Realized Sales Gross Margin: Realized Gross Profit/Net Sales
X6	Operating Profit Rate: Operating Income/Net Sales
X7	Pre-tax net Interest Rate: Pre-Tax Income/Net Sales
X8	After-tax net Interest Rate: Net Income/Net Sales
X9	Non-industry income and expenditure/revenue: Net Non-operating Income Ratio
X10	Continuous interest rate (after tax): Net Income-Exclude Disposal Gain or Loss/Net Sales
X11	Operating Expense Rate: Operating Expenses/Net Sales
X12	Research and development expense rate: (Research and Development Expenses)/Net Sales
X13	Cash flow rate: Cash Flow from Operating/Current Liabilities

X14	Interest-bearing debt interest rate: Interest-bearing Debt/Equity
X15	Tax rate (A): Effective Tax Rate
X16	Net Value Per Share (B): Book Value Per Share(B)
X17	Net Value Per Share (A): Book Value Per Share(A)
X18	Net Value Per Share (C): Book Value Per Share(C)
X19	Persistent EPS in the Last Four Seasons: EPS-Net Income
X20	Cash Flow Per Share
X21	Revenue Per Share (Yuan ¥): Sales Per Share
X22	Operating Profit Per Share (Yuan ¥): Operating Income Per Share
X23	Per Share Net profit before tax (Yuan ¥): Pretax Income Per Share
X24	Realized Sales Gross Profit Growth Rate
X25	Operating Profit Growth Rate: Operating Income Growth
X26	After-tax Net Profit Growth Rate: Net Income Growth
X27	Regular Net Profit Growth Rate: Continuing Operating Income after Tax Growth
X28	Continuous Net Profit Growth Rate: Net Income-Excluding Disposal Gain or Loss Growth
X29	Total Asset Growth Rate: Total Asset Growth
X30	Net Value Growth Rate: Total Equity Growth
X31	Total Asset Return Growth Rate Ratio: Return on Total Asset Growth
X32	Cash Reinvestment %: Cash Reinvestment Ratio
X33	Current Ratio
X34	Quick Ratio: Acid Test
X35	Interest Expense Ratio: Interest Expenses/Total Revenue
X36	Total debt/Total net worth: Total Liability/Equity Ratio
X37	Debt ratio %: Liability/Total Assets
X38	Net worth/Assets: Equity/Total Assets
X39	Long-term fund suitability ratio (A): (Long-term Liability+Equity)/Fixed Assets
X40	Borrowing dependency: Cost of Interest-bearing Debt
X41	Contingent liabilities/Net worth: Contingent Liability/Equity
X42	Operating profit/Paid-in capital: Operating Income/Capital
X43	Net profit before tax/Paid-in capital: Pretax Income/Capital
X44	Inventory and accounts receivable/Net value: (Inventory+Accounts Receivables)/Equity

X45	Total Asset Turnover
X46	Accounts Receivable Turnover
X47	Average Collection Days: Days Receivable Outstanding
X48	Inventory Turnover Rate (times)
X49	Fixed Assets Turnover Frequency
X50	Net Worth Turnover Rate (times): Equity Turnover
X51	Revenue per person: Sales Per Employee
X52	Operating profit per person: Operation Income Per Employee
X53	Allocation rate per person: Fixed Assets Per Employee
X54	Working Capital to Total Assets
X55	Quick Assets/Total Assets
X56	Current Assets/Total Assets
X57	Cash/Total Assets
X58	Quick Assets/Current Liability
X59	Cash/Current Liability
X60	Current Liability to Assets
X61	Operating Funds to Liability
X62	Inventory/Working Capital
X63	Inventory/Current Liability
X64	Current Liabilities/Liability
X65	Working Capital/Equity
X66	Current Liabilities/Equity
X67	Long-term Liability to Current Assets
X68	Retained Earnings to Total Assets
X69	Total income/Total expense
X70	Total expense/Assets
X71	Current Asset Turnover Rate: Current Assets to Sales
X72	Quick Asset Turnover Rate: Quick Assets to Sales
X73	Working capitcal Turnover Rate: Working Capital to Sales
X74	Cash Turnover Rate: Cash to Sales
X75	Cash Flow to Sales
X76	Fixed Assets to Assets
X77	Current Liability to Liability
X78	Current Liability to Equity
X79	Equity to Long-term Liability
X80	Cash Flow to Total Assets
X81	Cash Flow to Liability
X82	CFO to Assets
X83	Cash Flow to Equity
X84	Current Liability to Current Assets

X85	Liability-Assets Flag: 1 if Total Liability exceeds Total Assets, 0 otherwise
X86	Net Income to Total Assets
X87	Total assets to GNP price
X88	No-credit Interval
X89	Gross Profit to Sales
X90	Net Income to Stockholder's Equity
X91	Liability to Equity
X92	Degree of Financial Leverage (DFL)
X93	Interest Coverage Ratio (Interest expense to EBIT)
X94	Net Income Flag: 1 if Net Income is Negative for the last two years, 0 otherwise
X95	Equity to Liability

Compas	
target	Recidivism within two years?: Class label, 1 if yes, 0 if not
age	The age of a criminal
sex	The gender of a criminal: Male/Female
priors_count	Amount of prior arrests of a criminal
c_charge_degree	The charge degree: Felony/Misdemeanor (F/M)

Credit	
target	Class label, 1 if application for credit card accepted, 0 if not
reports	Number of major derogatory reports
age	Age n years plus twelfths of a year
income	Yearly income (divided by 10,000)
share	Ratio of monthly credit card expenditure to yearly income
expenditure	Average monthly credit card expenditure
owner	1 if owns their home, 0 if rent
selfempl	1 if self employed, 0 if not.
dependents	1 + number of dependents
months	Months living at current address
majorcards	Number of major credit cards held
active	Number of active credit accounts

Haberman	
target	Survival status: 0 = the patient survived 5 years or longer, 1 = the patient died within 5 year
age	Age of patient at time of operation
operation year	Patient's year of operation (year - 1900)
nodes	Number of positive axillary nodes detected

Water	
target	class attribute: 0 = water not safe, 1 = water is safe
aluminium	dangerous if greater than 2.8
ammonia	dangerous if greater than 32.5
arsenic	dangerous if greater than 0.01
barium	dangerous if greater than 2
cadmium	dangerous if greater than 0.005
chloramine	dangerous if greater than 4
chromium	dangerous if greater than 0.1
copper	dangerous if greater than 1.3
flouride	dangerous if greater than 1.5
bacteria	dangerous if greater than 0
viruses	dangerous if greater than 0
lead	dangerous if greater than 0.015
nitrates	dangerous if greater than 10
nitrites	dangerous if greater than 1
mercury	dangerous if greater than 0.002
perchlorate	dangerous if greater than 56
radium	dangerous if greater than 5
selenium	dangerous if greater than 0.5
silver	dangerous if greater than 0.1
uranium	dangerous if greater than 0.3

Admission	
target	Chance of Admit (ranging from 0 to 1)
GRE Score	The Graduate Record Examinations test (out of 340)
TOEFL Score	Test of English as a Foreign Language (out of 120)
University Rating	Rating of a university (out of 5)
SOP	Statement of Purpose (out of 5)
LOR	Letter of Recommendation Strength (out of 5)
CGPA	Undergraduate GPA (out of 10)
research	Research Experience (either 0 or 1)

AutoMPG	
target	Miles per galon (mpg)
cylinders	Amount of cylinders in car
displacement	Displacement of car
horsepower	Horsepower of car
weight	Weight of car
acceleration	Acceleration of car
model year	Model year of car - 1900
origin	Origin region of car
car name	Name of a car

Computer	
target	Price in US dollars of 486 PCs
SPEED	Clock speed in MHz
HARD DRIVE	Size of hard drive in MB
RAM	Size of Ram in in MB
SCREEN	Size of screen in inches
CD	Variable equals 1 if there is a CD-ROM present
MULTI	Variable equals 1 if a multimedia kit is included (speakers, sound card)
FIRM	Variable equals 1 if the manufacturer was a "premium" firm: IBM, COMPAQ
ADS	Number of 486 price listings for each month
TREND	Time trend indicating month starting from January of 1993 to November of 1995.

Kuiper	
target	Suggested retail price of the used 2005 GM car in excellent condition. The condition of a car can greatly affect price. All cars in this data set were less than one year old when priced and considered to be in excellent condition.
Mileage	Number of miles the car has been driven.
Make	Manufacturer of the car such as Saturn, Pontiac, and Chevrolet.
Model	Specific models for each car manufacturer such as Ion, Vibe, Cavalier Trim (of car): specific type of car model such as SE Sedan 4D, Quad Coupe 2D.
Type	Body type such as sedan, coupe, etc.
Cylinder	Number of cylinders in the engine.
Liter	A more specific measure of engine size.
Doors	Number of doors.
Cruise	Indicator variable representing whether the car has cruise control (1 = has cruise).
Sound	Indicator variable representing whether the car has upgraded speakers (1 = has upgraded).
Leather	Indicator variable representing whether the car has leather seats (1 = has leather).

Wages	
target	Average hourly earnings
educ	Years of education
exper	Years potential experience
tenure	Years with current employer
nonwhite	=1 If nonwhite
female	=1 If female
married	=1 If married
numdep	Number of dependents
smsa	=1 if live in SMSA
northcen	=1 if live in north central U.S
south	=1 if live in southern region
west	=1 if live in western region
construc	=1 if work in construc. indus.
ndurman	=1 if in nondur. manuf. indus.
trcommpu	=1 if in trans, commun, pub ut
trade	=1 if in wholesale or retail
services	=1 if in services indus.
profserv	=1 if in prof. serv. indus.
profocc	=1 if in profess. occupation
clerocc	=1 if in clerical occupation
servocc	=1 if in service occupation
lwage	log(wage)
expersq	$exper^2$
tenursq	$tenure^2$

Windsor	
target	Sale price of a house
lotsize	Lot size of a property in square feet
bedrooms	Number of bedrooms
bathrooms	Number of full bathrooms
stories	Number of stories excluding basement
driveway	Factor. Does the house have a driveway?
recreation	Factor. Does the house have a recreational room?
fullbase	Factor. Does the house have a full finished basement?
gasheat	Factor. Does the house use gas for hot water heating?
aircon	Factor. Is there central air conditioning?
garage	Number of garage places
prefer	Factor. Is the house located in the preferred neighborhood of the city?

Appendix C

Examples of generated classification / regression trees

The trees are traversed via depth-first traversal. In the following trees `EPSILON` stands for a very small positive number.

```
Node level) splitting rule to current node ; #data points ;
    impurity ; relative frequency of class 1 ; star (*) if node is
    leaf
0) root ; 150 ; 0.2498 ; 0.5133
1) 0.1264*X1+0.1031*X2+0.0956*X3+0.0928*X4+0.1061*X5 <= -0.0233;
    64 ; 0.1619 ; 0.2031 ; *
1) 0.1264*X1+0.1031*X2+0.0956*X3+0.0928*X4+0.1061*X5 > -0.0233 ;
    86 ; 0.1904 ; 0.7442
2) 0.0842*X1+0.108*X2+0.0743*X3+0.0442*X4 <= 0.1083 ; 48 ; 0.2287
    ; 0.6458
3) 0.0475*X1+0.0837*X2+0.1181*X3+0.1126*X4+0.0322*X6 <= 0.02 ; 21
    ; 0.2494 ; 0.4762
4) 0.1205*X1+0.1323*X2+0.1042*X3+0.0509*X5+0.0846*X6 <= 0.0607 ; 9
     ; 0.1728 ; 0.2222 ; *
4) 0.1205*X1+0.1323*X2+0.1042*X3+0.0509*X5+0.0846*X6 > 0.0607 ; 12
     ; 0.2222 ; 0.6667 ; *
3) 0.0475*X1+0.0837*X2+0.1181*X3+0.1126*X4+0.0322*X6 > 0.02 ; 27 ;
     0.1728 ; 0.7778 ; *
2) 0.0842*X1+0.108*X2+0.0743*X3+0.0442*X4 > 0.1083 ; 38 ; 0.1143 ;
     0.8684; *
```

```
Node level) splitting rule to current node ; #data points ;
    impurity ; relative frequency of class 1 ; star (*) if node is
    leaf
0) root ; 100 ; 6205973224.3298 ; 20025.6187
1) Type in ['Convertible', 'Coupe', 'Hatchback'] ; 35 ;
    2370238865.1284 ; 18764.3946
2) Type in ['Convertible'] ; 5 ; 44063450.4795 ; 10995.7287 ; *
2) Type not in ['Convertible'] ; 30 ; 1148271700.0074 ; 16396.0481
```

```
3) 0.0542*Mileage+4202.8796*Cylinder+584.7598*Cruise <= 27529.6603 ; 26 ; 290978985.4805 ; 14443.8844
4) Cylinder <= 5; 17 ; 34580285.1767 ; 12377.064
5) Type in ['Coupe'] ; 11 ; 16142097.2142 ; 13129.8272
6) Mileage <= 32064.5 ; 7 ; 7019706.4758 ; 12944.2903 ; *
6) Mileage > 32064.5 ; 4 ; 5370096.9844 ; 13446.4907 ; *
5) Type not in ['Coupe'] ; 6 ; 777520.4014 ; 13617.0104 ; *
4) Cylinder > 5; 9 ; 46608474.5121 ; 18347.8786
5) Type in ['Coupe'] ; 4 ; 33354160.6906 ; 18461.3818 ; *
5) Type not in ['Coupe'] ; 5 ; 1708994.5524 ; 19513.7689 ; *
3) 0.0542*Mileage+4202.8796*Cylinder+584.7598*Cruise > 27529.6604 ; 4 ; 114158827.4432 ; 19513.7689 ; *
1) Type not in ['Convertible', 'Coupe', 'Hatchback'] ; 65 ; 3750082029.7031 ; 20704.7393
2) Cylinder <= 7; 61 ; 2101841281.8652 ; 19432.3856
3) 0.1747*Mileage+8895.1079*Cruise+4237.4151*Leather <= 13944.2298 ; 22 ; 196234882.1082 ; 14080.34
4) 0.1394*Mileage+3261.5966*Cylinder+828.3051*Leather <= 22026.2558 ; 17 ; 50316215.9429 ; 12791.7983
5) Mileage <= 27010; 4 ; 9773060.5793 ; 19513.7689; *
5) Mileage > 27010; 13 ; 23669394.5928 ; 13344.4351
6) Mileage <= 34557.5 ; 9 ; 12875939.3045 ; 13522.2773
7) Mileage <= 29746.5 ; 4 ; 8915502.5278 ; 19513.7689 ; *
7) Mileage > 29746.5 ; 5 ; 3895821.3288 ; 19513.7689 ; *
6) Mileage > 34557.5 ; 4 ; 9868341.3307 ; 19513.76891 ; *
4) 0.1394*Mileage+3261.5966*Cylinder+828.3051*Leather > 22026.2558 ; 5 ; 21725255.5516 ; 19513.76891 ; *
3) 0.1747*Mileage+8895.1079*Cruise+4237.4151*Leather > 13944.2298 ; 39 ; 919945510.4052 ; 22451.4883
4) Type in ['Sedan'] ; 34 ; 601349566.5558 ; 21396.5966
5) Cylinder <= 5; 8 ; 180547783.2748 ; 24948.1217
6) Mileage <= 26682; 4 ; 156231333.766 ; 22611.9518 ; *
6) Mileage > 26682; 4 ; 3802321.2547 ; 22611.9518 ; *
5) Cylinder > 5; 26 ; 288846941.5599 ; 20303.8197
6) 0.1426*Mileage+2901.7214*Leather <= 5704.5447 ; 4 ; 5448729.6675 ; 22611.95187 ; *
6) 0.1426*Mileage+2901.7214*Leather > 5704.5447 ; 22 ; 225632420.0005 ; 20939.3955
7) Mileage <= 29287.5 ; 12 ; 123334958.7914 ; 20121.7373
8) Mileage <= 23747 ; 5 ; 56076020.5795 ; 22611.9518 ; *
8) Mileage > 23747 ; 7 ; 42883005.8794 ; 22611.9518 ; *
7) Mileage > 29287.5 ; 10 ; 84647346.4934 ; 21920.5853
8) Mileage <= 40195.5 ; 5 ; 63333732.4139 ; 22611.9518 ; *
8) Mileage > 40195.5 ; 5 ; 13374749.372 ; 22611.9518 ; *
4) Type not in ['Sedan'] ; 5 ; 23482322.6359 ; 22611.9518 ; *
2) Cylinder > 7; 4 ; 43522053.3229 ; 40108.1332 ; *
```

```
Node level) splitting rule to current node ; #data points ; impurity ; average target value ; star (*) if node is leaf
```

```
0) root ; 100 ; 70911881600; 69872
1) 3.1825*lotsize+82.8341*bedrooms+16807.1833*bathrooms+3716.2305*
   stories+3717.2915*driveway+9685.9931*recreation+7049.1971*
   fullbase+12264.6674*gasheat+17439.6493*aircon+5547.4447*garage
   +11752.355*prefer <= 81554.0288783031 ; 71 ; 17213889295.7746 ;
    57323.9437
2) 2.4082*lotsize+4168.9304*bedrooms+4469.1694*bathrooms
   -1869.1536*stories+1873.478*driveway+3900.1773*recreation
   +8125.5082*fullbase+19502.024*gasheat+9656.5426*aircon
   +2887.4921*garage+11156.1676*prefer <= 35181.9792 ; 38 ;
   3378450526.3158 ; 48484.2105
3) 2.7697*lotsize+5339.2773*bedrooms+1774.4207*bathrooms
   -2136.7676*stories-3154.9456*driveway+12598.1181*recreation
   +871.9266*fullbase-EPSILON*gasheat+14147.7296*aircon+3544.4235*
   garage+14338.0527*prefer <= 21075.2505 ; 9 ; 282500000 ;
   37666.6667
4) -0.1469*lotsize-6949.7062*bedrooms+EPSILON*bathrooms-6949.7062*
   stories-5211.3809*driveway+12185.7121*fullbase-2489.0212*garage
    <= -27446.9228 ; 4 ; 96750000 ; 57139.3707 ; *
4) -0.1469*lotsize-6949.7062*bedrooms+EPSILON*bathrooms-6949.7062*
   stories-5211.3809*driveway+12185.7121*fullbase-2489.0212*garage
    > -27446.9228 ; 5 ; 45300000 ; 57139.3707 ; *
3) 2.7697*lotsize+5339.2773*bedrooms+1774.4207*bathrooms
   -2136.7676*stories-3154.9456*driveway+12598.1181*recreation
   +871.9266*fullbase-EPSILON*gasheat+14147.7296*aircon+3544.4235*
   garage+14338.0527*prefer > 21075.2505 ; 29 ; 1715930344.8276 ;
   51841.3793
4) -0.76*lotsize+2316.1223*bedrooms-1872.7957*bathrooms-1780.0551*
   stories+3546.0938*driveway+10792.4654*recreation-5789.0335*
   fullbase-EPSILON*gasheat+972.0171*aircon+231.2239*garage
   +2045.4286*prefer <= 3927.0859 ; 24 ; 1328545000 ; 50225
5) -2.8891*lotsize-4031.9171*bedrooms-1969.0816*bathrooms
   -1279.107*stories+3452.5744*driveway-EPSILON*recreation
   -8480.0726*fullbase+EPSILON*gasheat-8435.8382*aircon-6571.3347*
   garage-6997.4673*prefer <= -27182.9530 ; 14 ; 673052142.8571 ;
   46635.7143
6) 10.1112*lotsize+18930.39*bedrooms+5799.0463*bathrooms
   -3223.7135*stories-9459.6992*driveway+20882.3991*fullbase
   +14556.0125*garage+43315.5646*prefer <= 105539.1111 ; 7 ;
   271500000 ; 57139.3707 ; *
6) 10.1112*lotsize+18930.39*bedrooms+5799.0463*bathrooms
   -3223.7135*stories-9459.6992*driveway+20882.3991*fullbase
   +14556.0125*garage+43315.5646*prefer > 105539.1111 ; 7 ;
   162094285.7143 ; 57139.3707 ; *
5) -2.8891*lotsize-4031.9171*bedrooms-1969.0816*bathrooms
   -1279.107*stories+3452.5744*driveway-EPSILON*recreation
   -8480.0726*fullbase+EPSILON*gasheat-8435.8382*aircon-6571.3347*
   garage-6997.4673*prefer > -27182.9530 ; 10 ; 222625000 ; 55250
6) 9.0874*lotsize+19493.9289*bedrooms+2178.8089*bathrooms
```

```
    +6861.7616*stories-2178.8089*driveway-EPSILON*recreation
    +18809.2898*aircon+21630.3363*garage+18628.0235*prefer <=
    102236.3882 ; 6 ; 9708333.3333 ; 57139.3707 ; *
15 6) 9.0874*lotsize+19493.9289*bedrooms+2178.8089*bathrooms
    +6861.7616*stories-2178.8089*driveway-EPSILON*recreation
    +18809.2898*aircon+21630.3363*garage+18628.0235*prefer >
    102236.3882 ; 4 ; 11250000 ; 57139.3707 ; *
16 4) -0.76*lotsize+2316.1223*bedrooms-1872.7957*bathrooms-1780.0551*
    stories+3546.0938*driveway+10792.4654*recreation-5789.0335*
    fullbase-EPSILON*gasheat+972.0171*aircon+231.2239*garage
    +2045.4286*prefer > 3927.0859 ; 5 ; 23700000 ; 57139.3707 ; *
17 2) 2.4082*lotsize+4168.9304*bedrooms+4469.1694*bathrooms
    -1869.1536*stories+1873.478*driveway+3900.1773*recreation
    +8125.5082*fullbase+19502.024*gasheat+9656.5426*aircon
    +2887.4921*garage+11156.1676*prefer > 35181.9792 ; 33 ;
    7446829696.9697 ; 67503.0303
18 3) 1.804*lotsize+1098.7594*bedrooms+5176.0107*bathrooms-111.6903*
    stories+8948.7435*driveway-8782.7751*recreation+12426.9128*
    fullbase+15108.4726*gasheat+5447.2623*aircon+1208.838*garage
    +5497.4776*prefer <= 44526.1943 ; 27 ; 2598085185.1852 ;
    64559.2593
19 4) 1.1901*lotsize-3519.5818*bedrooms+1997.9127*bathrooms
    +4227.1671*stories-6289.5327*driveway-3025.0043*recreation
    +102.8325*fullbase-11699.3256*gasheat+6319.9316*aircon
    -101.2794*garage+7096.9602*prefer <= 5208.1578 ; 20 ;
    1324445500 ; 60915
20 5) -1.2718*lotsize-2321.481*bedrooms-1915.4909*bathrooms
    -9168.5683*stories+7474.5438*driveway+7789.9035*recreation
    +5064.6062*fullbase-7474.5438*gasheat-2836.6256*aircon
    +522.7835*garage-7841.1007*prefer <= -26519.4095 ; 8 ;
    378340000 ; 54800
21 6) 15.8416*lotsize+705.9052*bedrooms+76493.0693*bathrooms
    +48188.1895*stories-36779.8444*driveway+EPSILON*recreation
    +36779.8444*fullbase+36779.8444*gasheat+89422.8076*aircon
    +4645.5092*garage+106888.0835*prefer <= 299022.5955 ; 4 ;
    33207500 ; 57139.3707 ; *
22 6) 15.8416*lotsize+705.9052*bedrooms+76493.0693*bathrooms
    +48188.1895*stories-36779.8444*driveway+EPSILON*recreation
    +36779.8444*fullbase+36779.8444*gasheat+89422.8076*aircon
    +4645.5092*garage+106888.0835*prefer > 299022.5955 ; 4 ;
    73687500; 57139.3707 ; *
23 5) -1.2718*lotsize-2321.481*bedrooms-1915.4909*bathrooms
    -9168.5683*stories+7474.5438*driveway+7789.9035*recreation
    +5064.6062*fullbase-7474.5438*gasheat-2836.6256*aircon
    +522.7835*garage-7841.1007*prefer > -26519.4095 ; 12 ;
    447529166.6667 ; 64991.6667
24 6) -0.6843*lotsize-EPSILON*bedrooms-9833.2559*bathrooms-1337.0514*
    stories+EPSILON*driveway+8145.8125*recreation+2438.9925*
    fullbase-8519.6831*aircon+1465.9379*garage-15061.7401*prefer <=
```

```
    -21183.0345 ; 4 ; 106127500; 57139.3707 ; *
25 6) -0.6843*lotsize-EPSILON*bedrooms-9833.2559*bathrooms-1337.0514*
   stories+EPSILON*driveway+8145.8125*recreation+2438.9925*
   fullbase-8519.6831*aircon+1465.9379*garage-15061.7401*prefer >
   -21183.0345 ; 8 ; 219000000; 67250
26 7) -1.5128*lotsize-EPSILON*bedrooms-8484.1328*bathrooms-4845.0262*
   stories+2394.8104*recreation-4845.0262*fullbase+1244.2963*
   aircon+6047.6369*garage+2394.8104*prefer <= -26026.1717 ; 4 ;
   50187500; 57139.3707 ; *
27 7) -1.5128*lotsize-EPSILON*bedrooms-8484.1328*bathrooms-4845.0262*
   stories+2394.8104*recreation-4845.0262*fullbase+1244.2963*
   aircon+6047.6369*garage+2394.8104*prefer > -26026.1717 ; 4 ;
   123687500; 57139.3707 ; *
28 4) 1.1901*lotsize-3519.5818*bedrooms+1997.9127*bathrooms
   +4227.1671*stories-6289.5327*driveway-3025.0043*recreation
   +102.8325*fullbase-11699.3256*gasheat+6319.9316*aircon
   -101.2794*garage+7096.9602*prefer > 5208.1578 ; 7 ;
   249134285.7143 ; 57139.3707 ; *
29 3) 1.804*lotsize+1098.7594*bedrooms+5176.0107*bathrooms-111.6903*
   stories+8948.7435*driveway-8782.7751*recreation+12426.9128*
   fullbase+15108.4726*gasheat+5447.2623*aircon+1208.838*garage
   +5497.4776*prefer > 44526.1943 ; 6 ; 3561875000; 57139.3707 ; *
30 1) 3.1825*lotsize+82.8341*bedrooms+16807.1833*bathrooms+3716.2305*
   stories+3717.2915*driveway+9685.9931*recreation+7049.1971*
   fullbase+12264.6674*gasheat+17439.6493*aircon+5547.4447*garage
   +11752.355*prefer > 81554.0288 ; 29 ; 15148978620.6896 ;
   100593.1034
31 2) 2.2366*lotsize-1635.8265*bedrooms+23509.875*bathrooms
   +4892.6759*stories-EPSILON*driveway+13705.5008*recreation
   +12977.5338*fullbase+4985.3512*gasheat+29799.388*aircon
   +10382.7418*garage+14683.9257*prefer <= 119293.8081 ; 19 ;
   3359325263.1579 ; 88984.2105
32 3) 1.5014*lotsize+5294.6804*bedrooms-5071.979*bathrooms-4584.0887*
   stories+EPSILON*driveway-19549.3876*recreation-4858.1512*
   fullbase-16126.1351*gasheat-897.4326*aircon-11300.0444*garage
   -13316.6288*prefer <= -17104.4824 ; 10 ; 300396000; 79720
33 4) 1.2897*lotsize-34205.1822*bedrooms+32190.2306*bathrooms
   +6976.3795*stories-EPSILON*driveway+7082.8788*recreation
   -2774.1083*fullbase+18430.9295*gasheat+23786.9585*aircon
   +18332.5549*garage+26950.5619*prefer <= 4599.5962 ; 6 ;
   76500000; 102319.4444 ; *
34 4) 1.2897*lotsize-34205.1822*bedrooms+32190.2306*bathrooms
   +6976.3795*stories-EPSILON*driveway+7082.8788*recreation
   -2774.1083*fullbase+18430.9295*gasheat+23786.9585*aircon
   +18332.5549*garage+26950.5619*prefer > 4599.5962 ; 4 ; 5010000;
    102319.4444 ; *
35 3) 1.5014*lotsize+5294.6804*bedrooms-5071.979*bathrooms-4584.0887*
   stories+EPSILON*driveway-19549.3876*recreation-4858.1512*
   fullbase-16126.1351*gasheat-897.4326*aircon-11300.0444*garage
```

```
    -13316.6288*prefer > -17104.4825 ; 9 ; 1247055555.5556 ;
    99277.7778
36 4) 14.9985*lotsize-2539.6581*bedrooms-3710.1878*bathrooms
    +417.2464*stories-EPSILON*driveway-19356.0399*recreation
    +14791.7436*fullbase+16169.2156*garage-6934.506*prefer <=
    83508.8786 ; 5 ; 209200000; 102319.4444 ; *
37 4) 14.9985*lotsize-2539.6581*bedrooms-3710.1878*bathrooms
    +417.2464*stories-EPSILON*driveway-19356.0399*recreation
    +14791.7436*fullbase+16169.2156*garage-6934.506*prefer >
    83508.8786 ; 4 ; 48500000; 102319.4444 ; *
38 2) 2.2366*lotsize-1635.8265*bedrooms+23509.875*bathrooms
    +4892.6759*stories-EPSILON*driveway+13705.5008*recreation
    +12977.5338*fullbase+4985.3512*gasheat+29799.388*aircon
    +10382.7418*garage+14683.9257*prefer > 119293.8081 ; 10 ;
    4364025000; 122650
39 3) -18.2486*lotsize-46302.8317*bedrooms+73005.506*bathrooms
    +44834.8191*stories-EPSILON*driveway+101632.9313*recreation
    +88459.8846*fullbase+EPSILON*gasheat+106828.5265*aircon
    +39818.8254*garage+108718.4059*prefer <= 295927.8972 ; 6 ;
    293208333.3333 ; 102319.4444 ; *
40 3) -18.2486*lotsize-46302.8317*bedrooms+73005.506*bathrooms
    +44834.8191*stories-EPSILON*driveway+101632.9313*recreation
    +88459.8846*fullbase+EPSILON*gasheat+106828.5265*aircon
    +39818.8254*garage+108718.4059*prefer > 295927.8972 ; 4 ;
    1638750000; 102319.4444 ; *
```

Appendix D

Tables of experiments

D.1 Performance tables for different metrics for real classification datasets

	local_global	not_local_global	local_not_global	not_local_not_global
Bankrupt	0.76087	0.790497	0.768473	0.721228
Compas	0.538350	0.635474	0.615066	0.455189
Credit	0.949612	0.949612	0.949612	0.949612
Haberman	0.474074	0.474074	0.369748	0.369748
Water	0.722162	0.666667	0.629200	0.450437

Table D.1: Experiments run for classification, statistic is F_1 and train set size is 50.

	local_global	not_local_global	local_not_global	not_local_not_global
Bankrupt	0.774359	0.751282	0.758974	0.720513
Compas	0.627333	0.630693	0.637023	0.536880
Credit	0.952030	0.952030	0.952030	0.952030
Haberman	0.642209	0.642209	0.584637	0.584637
Water	0.710259	0.500000	0.657835	0.574972

Table D.2: Experiments run for classification, statistic is auroc and train set size is 50.

	local_global	not_local_global	local_not_global	not_local_not_global
Bankrupt	0.809249	0.682836	0.739336	0.719388
Compas	0.646957	0.560151	0.587804	0.498892
Credit	1.000000	1.000000	1.000000	1.000000
Haberman	0.477612	0.477612	0.431373	0.431373
Water	0.693666	0.500000	0.686667	0.637113

Table D.3: Experiments run for classification, statistic is precision and train set size is 50.

	local_global	not_local_global	local_not_global	not_local_not_global
Bankrupt	0.712821	0.938462	0.8	0.723077
Compas	0.460967	0.734201	0.644981	0.418525
Credit	0.904059	0.904059	0.904059	0.904059
Haberman	0.470588	0.470588	0.323529	0.323529
Water	0.753100	1.000000	0.580609	0.348365

Table D.4: Experiments run for classification, statistic is recall and train set size is 50.

	local_global	not_local_global	local_not_global	not_local_not_global
Bankrupt	0.717949	0.666667	0.805031	0.804878
Compas	0.568073	0.000000	0.568073	0.622487
Credit	0.989858	0.989858	0.989858	0.989858
Haberman	0.431818	0.431818	0.436364	0.458333
Water	0.672112	0.666667	0.664516	0.626549

Table D.5: Experiments run for classification, statistic is F_1 and train set size is 100.

	local_global	not_local_global	local_not_global	not_local_not_global
Bankrupt	0.817647	0.5	0.817647	0.811765
Compas	0.652374	0.500000	0.652374	0.664112
Credit	0.989837	0.989837	0.989837	0.989837
Haberman	0.626370	0.626370	0.615533	0.637086
Water	0.674014	0.500000	0.638051	0.632831

Table D.6: Experiments run for classification, statistic is auroc and train set size is 100.

	local_global	not_local_global	local_not_global	not_local_not_global
Bankrupt	0.864865	0.5	0.864865	0.835443
Compas	0.689840	0.000000	0.689840	0.642644
Credit	0.987854	0.987854	0.987854	0.987854
Haberman	0.575758	0.575758	0.436364	0.536585
Water	0.676056	0.500000	0.619238	0.637455

Table D.7: Experiments run for classification, statistic is precision and train set size is 100.

	local_global	not_local_global	local_not_global	not_local_not_global
Bankrupt	0.752941	1	0.752941	0.776471
Compas	0.482845	0.000000	0.482845	0.603556
Credit	0.991870	0.991870	0.991870	0.991870
Haberman	0.345455	0.345455	0.436364	0.400000
Water	0.668213	1.000000	0.716937	0.616009

Table D.8: Experiments run for classification, statistic is recall and train set size is 100.

	local_global	not_local_global	local_not_global	not_local_not_global
Bankrupt	0.823129	0.823129	0.823129	0.823129
Compas	0.523738	0.523738	0.563289	0.560026
Credit	0.983982	0.983982	0.983982	0.983982
Haberman	0.000000	0.000000	0.000000	0.000000
Water	0.760615	0.760615	0.760615	0.760615

Table D.9: Experiments run for classification, statistic is F_1 and train set size is 150.

	local_global	not_local_global	local_not_global	not_local_not_global
Bankrupt	0.82069	0.82069	0.82069	0.820695
Compas	0.623655	0.623655	0.596377	0.614320
Credit	0.984163	0.984163	0.984163	0.984163
Haberman	0.500000	0.500000	0.500000	0.500000
Water	0.767622	0.767622	0.767622	0.767622

Table D.10: Experiments run for classification, statistic is auroc and train set size is 150.

	local_global	not_local_global	local_not_global	not_local_not_global
Bankrupt	0.812081	0.812081	0.812081	0.812081
Compas	0.654118	0.654118	0.552401	0.589379
Credit	0.995370	0.995370	0.995370	0.995370
Haberman	0.000000	0.000000	0.000000	0.000000
Water	0.784264	0.784264	0.784264	0.784264

Table D.11: Experiments run for classification, statistic is precision and train set size is 150.

	local_global	not_local_global	local_not_global	not_local_not_global
Bankrupt	0.834483	0.834483	0.834483	0.834483
Compas	0.436695	0.436695	0.574615	0.533459
Credit	0.972851	0.972851	0.972851	0.972851
Haberman	0.000000	0.000000	0.000000	0.000000
Water	0.738351	0.738351	0.738351	0.738351

Table D.12: Experiments run for classification, statistic is recall and train set size is 150.

D.2 Additional tables artificial datasets

Artificial datasets with two features		
Correlation	Number of generated labels	Relative frequency of label 1 per labelling
Positive	6	0.5245 0.481 0.522 0.5135 0.475 0.4965
Zero	6	0.484 0.5115 0.5815 0.429 0.6 0.486
Negative	10	0.539 0.4835 0.5875 0.5635 0.4685 0.4935 0.466 0.4635 0.572 0.587

Table D.13: Statistics of the generated labellings for datasets with two features.

Artificial datasets with six features		
Correlation	Number of generated labels	Relative frequency of label 1 per labelling
Positive	10	0.5205 0.489 0.4755 0.4715 0.494 0.522 0.4995 0.4885 0.4765 0.5145
Zero	10	0.4685 0.5185 0.4995 0.4985 0.4885 0.5105 0.515 0.5005 0.479 0.4975
Negative	10	0.5005 0.504 0.4855 0.5 0.511 0.4935 0.511 0.5005 0.472 0.4925

Table D.14: Statistics of the generated labellings for datasets with six features.

Artificial datasets with ten features		
Correlation	Number of generated labels	Relative frequency of label 1 per labelling
Positive	10	0.462 0.4925 0.4815 0.491 0.487 0.477 0.4745 0.477 0.491 0.4855
Zero	10	0.4845 0.4905 0.4935 0.512 0.4855 0.5145 0.52 0.482 0.4975 0.4915
Negative	10	0.5145 0.509 0.5135 0.52 0.4905 0.497 0.5015 0.4835 0.487 0.493

Table D.15: Statistics of the generated labellings for datasets with ten features.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.890428	0.884290	0.889444	0.883306
Zero	0.877482	0.877609	0.877482	0.877638
Negative	0.809724	0.839917	0.835204	0.839616

Table D.16: Experiments run for artificial data with 2 features, statistic is F_1 and train set size is 50.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.888277	0.882879	0.887416	0.882018
Zero	0.880090	0.876864	0.880090	0.878285
Negative	0.807845	0.825655	0.822381	0.826863

Table D.17: Experiments run for artificial data with 2 features, statistic is auroc and train set size is 50.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.881814	0.882494	0.881669	0.882348
Zero	0.917897	0.909537	0.917897	0.912844
Negative	0.807438	0.809792	0.810474	0.817985

Table D.18: Experiments run for artificial data with 2 features, statistic is precision and train set size is 50.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.903462	0.890642	0.901740	0.888920
Zero	0.842121	0.849159	0.842121	0.846344
Negative	0.827414	0.878173	0.865855	0.866763

Table D.19: Experiments run for artificial data with 2 features, statistic is recall and train set size is 50.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.903978	0.903978	0.903978	0.903978
Zero	0.896897	0.896897	0.895154	0.895154
Negative	0.854417	0.852407	0.851774	0.849765

Table D.20: Experiments run for artificial data with 2 features, statistic is F_1 and train set size is 100.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.905157	0.905157	0.905157	0.905157
Zero	0.892853	0.892853	0.891621	0.891621
Negative	0.840099	0.839960	0.838500	0.838361

Table D.21: Experiments run for artificial data with 2 features, statistic is auroc and train set size is 100.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.909476	0.909476	0.909476	0.909476
Zero	0.880573	0.880573	0.882340	0.882340
Negative	0.832704	0.836931	0.838054	0.842282

Table D.22: Experiments run for artificial data with 2 features, statistic is precision and train set size is 100.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.899214	0.899214	0.899214	0.899214
Zero	0.916332	0.916332	0.909796	0.909796
Negative	0.879540	0.870677	0.869834	0.860971

Table D.23: Experiments run for artificial data with 2 features, statistic is recall and train set size is 100.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.922758	0.918026	0.922758	0.918026
Zero	0.899106	0.899106	0.899106	0.899106
Negative	0.845936	0.844081	0.847037	0.845182

Table D.24: Experiments run for artificial data with 2 features, statistic is F_1 and train set size is 150.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.920353	0.916294	0.920353	0.916294
Zero	0.895284	0.895284	0.895284	0.895284
Negative	0.839808	0.838582	0.841290	0.840064

Table D.25: Experiments run for artificial data with 2 features, statistic is auroc and train set size is 150.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.913770	0.916483	0.913770	0.916483
Zero	0.889251	0.889251	0.889251	0.889251
Negative	0.844009	0.847009	0.848720	0.851721

Table D.26: Experiments run for artificial data with 2 features, statistic is precision and train set size is 150.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.857481	0.725310	0.857481	0.835226
Zero	0.630846	0.439255	0.621969	0.598894
Negative	0.561449	0.370740	0.576655	0.574359

Table D.27: Experiments run for artificial data with 6 features, statistic is F_1 and train set size is 50.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.855551	0.741006	0.855551	0.835937
Zero	0.643776	0.552564	0.640858	0.628848
Negative	0.580855	0.531392	0.582334	0.571806

Table D.28: Experiments run for artificial data with 6 features, statistic is auroc and train set size is 50.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.849366	0.682753	0.849366	0.832623
Zero	0.656738	0.418768	0.657063	0.654743
Negative	0.582782	0.325225	0.578106	0.564232

Table D.29: Experiments run for artificial data with 6 features, statistic is precision and train set size is 50.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.868982	0.802974	0.868982	0.840766
Zero	0.627364	0.521662	0.607954	0.569975
Negative	0.576410	0.461988	0.604655	0.596944

Table D.30: Experiments run for artificial data with 6 features, statistic is recall and train set size is 50.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.874451	0.492747	0.873855	0.858632
Zero	0.671988	0.543841	0.671988	0.668032
Negative	0.611436	0.437192	0.588860	0.579953

Table D.31: Experiments run for artificial data with 6 features, statistic is F_1 and train set size is 100.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.872611	0.675118	0.871791	0.857790
Zero	0.655514	0.628316	0.655514	0.657753
Negative	0.583815	0.550818	0.589498	0.586856

Table D.32: Experiments run for artificial data with 6 features, statistic is auroc and train set size is 100.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.857944	0.467938	0.854310	0.845174
Zero	0.635877	0.512414	0.635877	0.638829
Negative	0.574429	0.392241	0.584566	0.581924

Table D.33: Experiments run for artificial data with 6 features, statistic is precision and train set size is 100.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.893374	0.534786	0.895572	0.874019
Zero	0.727473	0.595928	0.727473	0.710528
Negative	0.692980	0.527421	0.614174	0.597424

Table D.34: Experiments run for artificial data with 6 features, statistic is recall and train set size is 100.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.873336	0.521270	0.873336	0.859819
Zero	0.683822	0.671545	0.679722	0.671329
Negative	0.585609	0.466756	0.576639	0.552933

Table D.35: Experiments run for artificial data with 6 features, statistic is F_1 and train set size is 150.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.872490	0.601872	0.872490	0.860114
Zero	0.668186	0.647427	0.667171	0.663315
Negative	0.585943	0.553026	0.582121	0.574823

Table D.36: Experiments run for artificial data with 6 features, statistic is auroc and train set size is 150.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.863846	0.455900	0.863846	0.856395
Zero	0.647539	0.634178	0.649228	0.649252
Negative	0.581792	0.455398	0.578767	0.581024

Table D.37: Experiments run for artificial data with 6 features, statistic is precision and train set size is 150.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.884908	0.650676	0.884908	0.864145
Zero	0.736653	0.747779	0.725849	0.715986
Negative	0.601084	0.537467	0.583633	0.565478

Table D.38: Experiments run for artificial data with 6 features, statistic is recall and train set size is 150.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.833968	0.630298	0.834298	0.811984
Zero	0.495516	0.464085	0.508867	0.485212
Negative	0.379014	0.459481	0.460310	0.372538

Table D.39: Experiments run for artificial data with 10 features, statistic is F_1 and train set size is 50.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.838384	0.700551	0.839403	0.813447
Zero	0.509944	0.500000	0.508476	0.501601
Negative	0.496340	0.497039	0.492782	0.491526

Table D.40: Experiments run for artificial data with 10 features, statistic is auroc and train set size is 50.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.832854	0.588033	0.837537	0.792295
Zero	0.507698	0.347179	0.504867	0.501615
Negative	0.439086	0.349752	0.440924	0.343914

Table D.41: Experiments run for artificial data with 10 features, statistic is precision and train set size is 50.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.838860	0.709040	0.834586	0.835375
Zero	0.515778	0.700000	0.542482	0.486483
Negative	0.388450	0.673374	0.502682	0.426002

Table D.42: Experiments run for artificial data with 10 features, statistic is recall and train set size is 50.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.852164	0.394358	0.852164	0.819812
Zero	0.378082	0.267781	0.424172	0.350794
Negative	0.506166	0.388598	0.486871	0.498882

Table D.43: Experiments run for artificial data with 10 features, statistic is F_1 and train set size is 100.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.854473	0.630786	0.854473	0.825782
Zero	0.511676	0.500000	0.510744	0.508858
Negative	0.493832	0.500350	0.495367	0.497458

Table D.44: Experiments run for artificial data with 10 features, statistic is auroc and train set size is 100.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.842115	0.381064	0.842115	0.829617
Zero	0.411574	0.201263	0.409939	0.365426
Negative	0.444419	0.303582	0.444344	0.446496

Table D.45: Experiments run for artificial data with 10 features, statistic is precision and train set size is 100.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.866427	0.427268	0.866427	0.813757
Zero	0.409441	0.400000	0.451727	0.352693
Negative	0.621630	0.555187	0.572387	0.590394

Table D.46: Experiments run for artificial data with 10 features, statistic is recall and train set size is 100.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.84931	0.234264	0.849310	0.815415
Zero	0.40134	0.264667	0.436167	0.456217
Negative	0.39190	0.337282	0.403522	0.343964

Table D.47: Experiments run for artificial data with 10 features, statistic is F_1 and train set size is 150.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.854933	0.567188	0.854933	0.824456
Zero	0.512552	0.505300	0.514321	0.509669
Negative	0.497685	0.500000	0.502253	0.499071

Table D.48: Experiments run for artificial data with 10 features, statistic is auroc and train set size is 150.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.855450	0.209117	0.855450	0.832026
Zero	0.463931	0.259369	0.463657	0.454862
Negative	0.349469	0.254486	0.355187	0.303426

Table D.49: Experiments run for artificial data with 10 features, statistic is precision and train set size is 150.

	local_global	not_local_global	local_not_global	not_local_not_global
Positive	0.845005	0.277258	0.845005	0.804061
Zero	0.387933	0.339951	0.437108	0.493395
Negative	0.478059	0.500000	0.493097	0.414265

Table D.50: Experiments run for artificial data with 10 features, statistic is recall and train set size is 150.