TRINARY DECISION TREES FOR HANDLING MISSING DATA

HENNING ZAKRISSON
DEPARTMENT OF MATHEMATICS, STOCKHOLM UNIVERSITY

Abstract. This paper introduces the Trinary decision tree, an algorithm designed to improve the handling of missing data in decision tree regressors and classifiers. Unlike other approaches, the Trinary decision tree does not assume that missing values contain any information about the response. Both theoretical calculations on estimator bias and numerical illustrations using real data sets are presented to compare its performance with established algorithms in different missing data scenarios (Missing Completely at Random (MCAR), and Informative Missingness (IM)). Notably, the Trinary tree outperforms its peers in MCAR settings, especially when data is only missing out-of-sample, while lacking behind in IM settings. A hybrid model, the TrinaryMIA tree, which combines the Trinary tree and the Missing In Attributes (MIA) approach, shows robust performance in all types of missingness. Despite the potential drawback of slower training speed, the Trinary tree offers a promising and more accurate method of handling missing data in decision tree algorithms.
Keywords: Missing data, Decision trees, Regularization

1. Introduction

Missing values are prevalent in real data. As noted by e.g. Nijman et al. (2022), this is often not handled or mentioned in machine learning applications in a satisfactory way. Classification and Regression Trees (CART), as defined by Breiman et al. (1984) provide numerous ways to handle missing values in covariates. Since CARTs are the foundation of many increasingly popular machine learning algorithms such as Gradient Boosting Machines (GBMs) (Friedman 2001), Random Forests (Ho 1995), and XGBoost (Chen et al. 2015), they are still relevant today. But the proposed methods of handling missing data come with drawbacks.

The simplest way to handle missing values when training a tree is to simply ignore them by discarding data points with any missing feature. This of course means losing potentially useful information, and is not an option when predicting using data with missing values. The perhaps second-simplest method is using the *majority rule*, where data points with missing values are assigned to the category in a tree split with the largest amount of data in training. Another method is presented by Twala et al. (2008), the Missing In Attributes (MIA) algorithm. MIA assigns data points with missing covariates to the category that minimizes the loss for the training data. This is similar, but not identical, to assigning missing values an own category in a categorical feature. Quinlan (1993) introduces the C4.5 algorithm for decision trees, and with that a weighted probabilistic strategy for missing value-handling, henceforth referred to as Fractional Case - FC. In FC, a data point with a missing value in a split is assigned a weight of membership in both categories of a binary splot, with the weight depending on the distribution of the observable data in the node. For out-of-sample data, the weights for all terminal nodes are calculated and the prediction is given as a weighted average. Breiman et al. (1984) proposes using so-called *surrogate splits* in order to find other covariates on which the data points which lack the relevant observation can be split to form similar splits. This requires that there are no missing values in the surrogate covariate - or that a secondary surrogate variable is found in its place.

Cons of these methods include losing potentially useful information (discarding data, FC), assuming there is always information in missingness (MIA), requiring missing values in the training data to be able to handle missing values in out-of-sample prediction (MIA, surrogate splits) or losing interpretability (FC).

The trinary decision tree for missing value handling (henceforth Trinary tree) introduced in this paper has four important attributes:

- It does not assume that missing data points contain any information about the response
- It can handle missing values in predictions even if it was trained on a data set with no missing data
- It maintains the interpretability of a standard decision tree
- It produces locally unbiased estimators of tree node values - which the other algorithms do not necessarily do

The first three are apparent from the algorithm, which is presented in Section 2, whereas the fourth one is proven in Section 3. In Section 4, the algorithm is tested against its peers with real data sets.

2. The Trinary tree

Consider a loss function $\mathcal{L}((y_i)_{i\in\mathcal{I}}, \delta)$, where $\mathcal{I}$ is an index set and δ is a parameter. For regression problems, δ is a real number, and for classification problems, δ is a probability vector. In this paper, two loss functions will be considered: the sum of squared errors (SSE) for regression and the point-wise cross-entropy for classification. These are defined as

$$\mathcal{L}^{\text{SSE}}((y_i)_{i\in\mathcal{I}}, \delta) = \sum_{i\in\mathcal{I}} (y_i - \delta)^2, \tag{1}$$

and

$$\mathcal{L}^{\mathrm{XE}}((y_i)_{i\in\mathcal{I}},\delta)=\sum_{i\in\mathcal{I}}\log(\delta_{y_i}) \tag{2}$$

respectively. Let $\delta_{\mathcal{I}}$ denote the minimizer of $\mathcal{L}((y_i)_{i\in\mathcal{I}},\delta)$, i.e.

$$\delta_{\mathcal{I}}=\arg\min_{\delta}\mathcal{L}((y_i)_{i\in\mathcal{I}},\delta) \tag{3}$$

For both SSE and point-wise cross-entropy, $\delta_{\mathcal{I}}$ has a closed form solution, namely

$$\delta_{\mathcal{I}}^{\mathrm{SSE}}=\frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}}y_i \tag{4}$$

and

$$\left(\delta_{\mathcal{I}}^{\mathrm{XE}}\right)_k=\frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}}\mathbb{1}_{\{y_i\}=k},\quad k=1,\dots,K \tag{5}$$

where K is the number of classes in the classification problem. A binary decision tree is generally fitted greedily by minimizing the loss function at each so called *node* separately, starting with the *root node* containing all data points. For data set $(y_i,x_i)_{i=1}^n$, where $y_i\in\mathcal{Y}$ is the response and $x_{ij}\in\mathcal{X}_j$, $j=1,\dots,p$, is a covariate, this means finding a combination of a covariate j and covariate subspaces $\mathcal{X}_j^l,\mathcal{X}_j^r\subset\mathcal{X}_j$ that minimize

$$\mathcal{L}((y_i)_{i\in\mathcal{I}_{jl}},\delta_{\mathcal{I}_{jl}})+\mathcal{L}((y_i)_{i\in\mathcal{I}_{jr}},\delta_{\mathcal{I}_{jr}}), \tag{6}$$

where

$$\mathcal{I}_{jl}=\{i\in\mathcal{I}:x_{ij}\in\mathcal{X}_j^l\},\qquad \mathcal{I}_{jr}=\{i\in\mathcal{I}:x_{ij}\in\mathcal{X}_j^r\}, \tag{7}$$

such that $\mathcal{X}_j^l\cup\mathcal{X}_j^r=\mathcal{X}_j$. In the case where $\mathcal{X}\in\mathbb{R}$, $\mathcal{X}_j^l$ and $\mathcal{X}_j^r$ are constrained to be continuous intervals. After finding the optimal split, the procedure is repeated for the so called *daughter nodes*, i.e. the nodes that contain the data points from the two sides of the split. The procedure is generally continued until reaching some stopping criterion, such as a maximum tree depth. The nodes that are not split are called *terminal nodes*.

Data points where the chosen splitting covariate is missing can be handled in a number of ways. The *MIA* strategy assigns them to the daughter node that provides the largest reduction in the loss function (see Algorithm 2 in Appendix A). The *majority* strategy assigns them to the daughter node with the largest amount of data (see Algorithm 3 in Appendix A). The *FC* strategy assigns them to both daughter nodes with weights depending on the distribution of the observable data in the node (see Algorithm 4 in Appendix A).

In contrast, the *Trinary* strategy assigns them to a third daughter node, which changes the function to minimize in (6) to

$$\mathcal{L}((y_i)_{i\in\mathcal{I}_{jl}},\delta_{\mathcal{I}_{jl}})+\mathcal{L}((y_i)_{i\in\mathcal{I}_{jr}},\delta_{\mathcal{I}_{jr}})+\mathcal{L}((y_i)_{i\in\mathcal{I}_{jm}},\delta_{\mathcal{I}}), \tag{8}$$

where $\mathcal{I}_{jm}=\{i\in\mathcal{I}:x_{ij}\text{ missing}\}$. Note that $\delta_{\mathcal{I}}$ in the third term is the minimizer of the loss function over the entire data set of the node. This means that the third term evaluates the loss of the points that are not assigned to the left or right nodes as if it retained the parameter estimate of the mother node. After finding a split, the procedure is repeated for all three daughter nodes. For the third node, the entire data set is used for further splitting, but omitting the splitting covariate. Thus, the third node will grow further by first splitting on the second-best covariate, then continue to grow. The third node is considered to be at the same depth level as the mother node, since the data set has not been split.

The point of the third node is to avoid making assumptions about the missing data. By not assigning the missing data to either of the standard daughter nodes, the Trinary tree does not contaminate the δ estimates for the standard nodes with data points that do not belong there. Instead, the Trinary tree uses the entire data set to estimate δ for the third node, as a way to regularize predictions when in uncertainty about important covariates.

The Trinary tree training algorithm is summarized in Algorithm 1. A visualization of a Trinary tree with depth 1 is shown in Figure 1.

Algorithm 1 Trinary tree training algorithm

Let

- $(y_i,x_i)_{i\in\mathcal{I}}$, where $y_i\in\mathcal{Y}$, $x_{ij}\in\mathcal{X}_j$, $j=1,\dots,p$, be the training data
- $\mathcal{L}((y_i)_{i\in\mathcal{I}},\delta)$ be the loss given parameter δ
- $\delta_{\mathcal{I}}$ be the minimizing parameter of $\mathcal{L}((y_i)_{i\in\mathcal{I}},\delta)$
- $\mathcal{I}_{jl}=\{i\in\mathcal{I}:x_{ij}\in\mathcal{X}_j^l\}$, $\mathcal{I}_{jr}=\{i\in\mathcal{I}:x_{ij}\in\mathcal{X}_j^r\}$
- $\mathcal{I}_{jm}=\{i:x_{ij}\text{ missing}\}$
- $d_{\max}$ be the maximum depth
- n be the minimum number of samples per node

Define training function $\mathcal{T}\left((y_i,x_i)_{i\in\mathcal{I}},d\right)\to h(x)$:

If $d=d_{\max}$ or $|\mathcal{I}|=n$:

Output

$$h(x)=\delta_{\mathcal{I}}$$

Else:

Fit

Find j, $\mathcal{X}_j^l$, and $\mathcal{X}_j^r$ that minimize

$$\mathcal{L}\left((y_i)_{i\in\mathcal{I}_{jl}},\delta_{\mathcal{I}_{jl}}\right)+\mathcal{L}\left((y_i)_{i\in\mathcal{I}_{jr}},\delta_{\mathcal{I}_{jr}}\right)+\mathcal{L}\left((y_i)_{i\in\mathcal{I}_{jm}},\delta_{\mathcal{I}}\right)$$

such that $|\mathcal{I}_l|\geq n$, $|\mathcal{I}_r|\geq n$, $\mathcal{X}_j^l\cup\mathcal{X}_j^r=\mathcal{X}_j$

Grow

$$h_l(x)=\mathcal{T}\left((y_i,x_i)_{\mathcal{I}_l},d+1\right)$$
$$h_r(x)=\mathcal{T}\left((y_i,x_i)_{\mathcal{I}_r},d+1\right)$$
$$h_m(x)=\mathcal{T}\left((y_i,x_i)_{\mathcal{I}},d\right)$$

Output

$$h(x)=\begin{cases}h_l(x), & x_{\cdot j}\in\mathcal{X}_j^l\\ h_r(x), & x_{\cdot j}\in\mathcal{X}_j^r\\ h_m(x), & x_{\cdot j}\text{ missing}\end{cases}$$

3. Tree-fitting estimate bias

In order to illustrate how the Trinary tree might be preferable to the other methods, let us examine a simple example where the non-trinary methods estimators are locally biased. Consider the data set $\mathcal{D}=(X_i,Y_i)_{i=1}^n$, where $X_i\in\mathcal{X}$ is the covariate and $Y_i\in\mathbb{R}$ is the response.

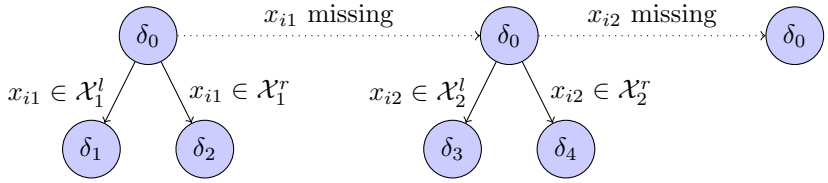

FIGURE 1. Visualization of a Trinary tree with depth 1 for a covariate with $p = 2$ dimensions. Note that since the third node is considered to be at the same depth level as the root node, an additional split is made. Since the best performing split covariate $j = 1$ is no longer available, the second-best split covariate $j = 2$ is used for the second split. Since that covariate could also be missing, the third node has its own third daughter node. Since there are no further covariates to split on, that node is a terminal node.

Let the expected value of Y follow a tree structure, i.e. let $\mathbb{E}[Y|X = x] = h(x)$ where

$$h(x) = \begin{cases} a, & x \in \mathcal{X}^l \\ b, & x \in \mathcal{X}^r \end{cases}$$

where $\mathcal{X}^l \cup \mathcal{X}^r = \mathcal{X}$, $a < b$ and $\mathbb{P}(X \in \mathcal{X}^r) = p$, $0 < p < 1$. Then, consider a censored dataset $\widetilde{\mathcal{D}} = (\tilde{X}_i, Y_i)_{i=1}^n$ where

$$\tilde{X}_i = \begin{cases} \texttt{nan}, & \text{with probability } q \\ X_i, & \text{with probability } 1 - q \end{cases}$$

where `nan` corresponds to a missing value, $q > 0$. Now, consider fitting trees to this for an SSE loss function, using the Majority, MIA and FC algorithms respectively, and examine their estimates of a. For brevity, let $\mathcal{I}_l$ and $\mathcal{I}_r$ be defined as in (7) and introduce index sets

$$\mathcal{I}^o = \left\{ i : \tilde{X}_i = X_i \right\}, \qquad \mathcal{I}^m = \left\{ i : \tilde{X}_i = \texttt{nan} \right\},$$

as well as the intersections

$$\begin{aligned} \mathcal{I}_l^o &= \mathcal{I}^o \cap \mathcal{I}_l, & \mathcal{I}_r^o &= \mathcal{I}^o \cap \mathcal{I}_r, \\ \mathcal{I}_l^m &= \mathcal{I}^o \cap \mathcal{I}_l, & \mathcal{I}_r^m &= \mathcal{I}^m \cap \mathcal{I}_r. \end{aligned}$$

For the majority rule algorithm, the estimate of a is

$$\hat{a}_{\text{Maj}} = \begin{cases} \frac{1}{|\mathcal{I}_l^o|} \sum\limits_{i \in \mathcal{I}_l^o} Y_i, & \text{if } |\mathcal{I}_l^o| > |\mathcal{I}_r^o|, \\ \frac{1}{|\mathcal{I}_l^o| + |\mathcal{I}^m|} \sum\limits_{i \in \mathcal{I}_l^o \cup \mathcal{I}^m} Y_i, & \text{else.} \end{cases}$$

First note that

$$\mathbb{E}[\hat{a}_{\text{Maj}} | \, |\mathcal{I}_l^o| \leq |\mathcal{I}_r^o|] = a.$$

Also,

$$\begin{aligned} &\mathbb{E}\left[\hat{a}_{\text{Maj}} | \, |\mathcal{I}_l^o| > |\mathcal{I}_r^o|\right] \\ &= \mathbb{E}\left[\frac{1}{|\mathcal{I}_l^o| + |\mathcal{I}_l^m| + |\mathcal{I}_r^m|} \left(\sum_{i \in \mathcal{I}_l^o} Y_i + \sum_{i \in \mathcal{I}_l^m} Y_i + \sum_{i \in \mathcal{I}_r^m} Y_i \right) \right] \\ &= \mathbb{E}\left[\frac{1}{|\mathcal{I}_l^o| + |\mathcal{I}_l^m| + |\mathcal{I}_r^m|} \left(a \, |\mathcal{I}_l^o| + a \, |\mathcal{I}_l^m| + b \, |\mathcal{I}_r^m| \right) \right] \\ &\geq \frac{1}{\mathbb{E}\left[|\mathcal{I}_l^o| + |\mathcal{I}_l^m| + |\mathcal{I}_r^m| \right]} \mathbb{E}\left[a \, |\mathcal{I}_l^o| + a \, |\mathcal{I}_l^m| + b \, |\mathcal{I}_r^m| \right] \\ &= a + \frac{pq}{1 - p + pq} (b - a) \end{aligned}$$

where the concavity of the function

$$g(x, y, z) = \frac{x}{x + y + z}$$

is used for the inequality. Then, introduce $\kappa = \mathbb{P}\left(|\mathcal{I}_l^o| \leq |\mathcal{I}_r^o| \right)$, and note that

$$\begin{aligned} \mathbb{E}\left[\hat{a}_{\text{Maj}}\right] &= \kappa \mathbb{E}[\hat{a}_{\text{Maj}} | \, |\mathcal{I}_l^o| \leq |\mathcal{I}_r^o|] + (1 - \kappa) \, \mathbb{E}[\hat{a}_{\text{Maj}} | \, |\mathcal{I}_l^o| > |\mathcal{I}_r^o|] \\ &\geq \kappa a + (1 - \kappa) \left(a + \frac{pq}{1 - p + pq} (b - a) \right) \\ &= a + \frac{(1 - \kappa) \, pq}{1 - p + pq} (b - a) \\ &> a \end{aligned}$$

The proof for the MIA strategy is identical with the only change that κ then means the probability that the loss is lower if the missing values are assigned to the left node. For the Fractional Case strategy the estimate of parameter a has expected value

$$\begin{aligned} \mathbb{E}\left[\hat{a}_{\text{FC}}\right] &= \mathbb{E}\left[\frac{1}{\sum\limits_{i=1}^n w_i^l} \sum_{i=1}^n w_i^l Y_i \right] \\ &\geq \frac{1}{\mathbb{E}\left[\sum\limits_{i=1}^n w_i^l \right]} \mathbb{E}\left[\sum_{i=1}^n w_i^l Y_i \right] \\ &= a + pq \, (b - a) \\ &> a. \end{aligned}$$

where it can also be shown that in order for $\mathbb{E}[\hat{Y}_{\text{FC}}] = \mathbb{E}[Y]$ to hold it is required that $\mathbb{E}[\hat{b}_{\text{FC}}] < b$. Finally, for the Trinary tree, see that

$$\mathbb{E}\left[\hat{a}_{\text{Tri}}\right] = \mathbb{E}\left[\frac{1}{|\mathcal{I}_l^o|} \sum_{i \in \mathcal{I}_l^o} Y_i \right] = a.$$

4. Numerical illustration

In order to illustrate the benefits of the Trinary tree, data with increasing missingness is created for the data sets in Table 4. All the data sets are available online. The data sets have been chosen in order to provide a wide array of applications, and varying characteristics of the data. The performance of the individual data sets will not be evaluated specifically in this paper, but rather the performance of the algorithms on the data sets as a

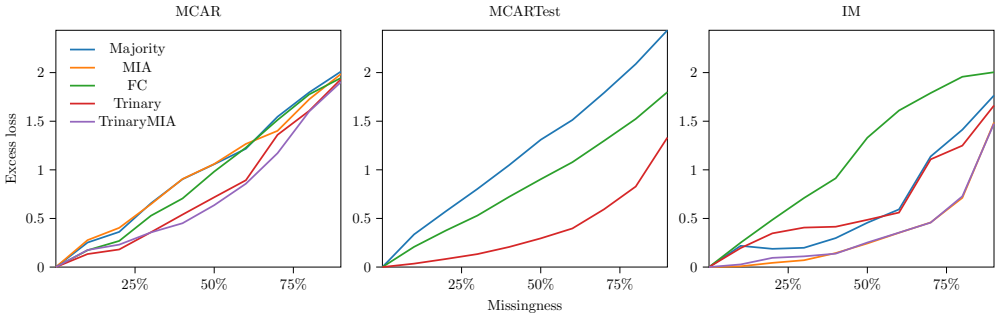

FIGURE 2. Average excess loss per missingness ratio for the tree algorithms in different kinds of missingness on all data sets

whole. The data sets have been minimally pre-processed, e.g. by removing any potential missing values. This is done in order to have full control over the missingness. The tree depth is first tuned by performing 10-fold cross validation on the full data set, with a maximum tree depth set to 5. For the classification problems, the folds are stratified so that the relative class frequency is equal in every fold. The dataset is then censored, i.e. values are replaced with missing values, (with missingness q% ranging from 0% to 90%) in three different ways.

MCAR q% of the data is removed from all features in the training and test set, completely at random.

MCARTest q% of the data is removed from all features in the test set, completely at random. The training set has no missing values.

IM q% of the data is removed from all features in the training and test set. For numerical features, the largest values are removed first. For categorical ones, they are removed on a category-by-category basis.

Thereafter, the four different tree algorithms (Majority, MIA, FC, and Trinary) are trained and evaluated on the 10 folds. Additionally, a fifth tree algorithm, denoted TrinaryMIA is evaluated. This is an amalgamation of Trinary and MIA that in every node evaluates whether a MIA-style or Trinary Tree style split reduces the training loss the most and then picks the one that does.

TABLE 1. Data sets

type	Name	size	features
regression	AutoMPG	392	8
	Black Friday	550,068	6
	Cement	1,030	9
	Life Expectancy	138	17
classification	Titanic	712	7
	Lymphography	142	19
	Boston Housing	506	14
	Wheat seeds	199	8

First the total loss is calculated for the entire data set with no missing values. Then the total loss is calculated for the data set with increasing missingness, and the excess loss (i.e. the loss divided by the loss in the case with no missing values) is calculated. The average excess loss over all data sets is presented in Figure 2. Since the MIA and majority strategies are identical in cases where there is no missing data in the training data, MIA is omitted from the middle figure. The same applies to Trinary and TrinaryMIA trees.

As can be seen, the TrinaryMIA tree is the best performer in the MCAR case, followed by the Trinary tree. For higher levels of missingness all algorithms seem to perform almost equally bad. In the MCARTest case, the Trinary tree is the best performer, followed by the FC tree. For the IM case, the TrinaryMIA and MIA trees perform very similarly, and are the best performers. The performance of MIA in this setting is expected, but it seems like the TrinaryMIA tree is able to find the appropriate splits as well.

5. Concluding remarks

It is clear both from the bias calculations and the numerical illustration that the Trinary tree has benefits over its peers in MCAR settings. Especially, the performance of the algorithm in the case where data is only missing in the test set is noteworthy. It is however important to remember that assuming that missingness contains no information is an assumption in itself - seen by the less impressive performance in the IM test. This drawback seems to be easily overcome by the TrinaryMIA tree, which maintains performance in all types of missingness. Surprisingly, the TrinaryMIA tree also outperforms the Trinary tree in the MCAR case, however not by a large margin.

The potential of using the Trinary tree algorithm as a weak learner in more powerful machine learning algorithms, such as a GBM, is an interesting topic, since the reguralization and missing value-handling would then be inherited by the ensemble model.

A drawback for the Trinary tree, is that for large data sets (especially covariate data sets with many features and categorical features with high cardinality) or deep trees, the training speed can suffer. For shallow trees and data sets with a limited number of covariates, the speed is however on par with the other methods. It should also be noted that a large part of the tree training can be parallelized, since nodes can be trained independently of each other when using standard greedy splitting. Also, TrinaryMIA training is often faster since it, if there is information in missingness in the training data, will grow fewer nodes than a standard Trinary tree.

Acknowledgements

The author would like to thank Mathias Lindholm for his valuable comments and suggestions. The trinary tree algorithm is available in the `python` package `trinary-tree` on `PyPI`, with the source code available at `https://github.com/henningzakrisson/trinary-tree`. Note that the package is a proof-of-concept implementation and is not optimized for speed.

References

Breiman, L., Friedman, J., Stone, C. & Olshen, R. (1984), *Classification and Regression Trees*, Taylor & Francis.

Chen, T., He, T., Benesty, M., Khotilovich, V., Tang, Y., Cho, H., Chen, K., Mitchell, R., Cano, I., Zhou, T. et al. (2015), 'Xgboost: extreme gradient boosting', *R package version 0.4-2* **1**(4), 1–4.

Friedman, J. H. (2001), 'Greedy function approximation: a gradient boosting machine', *Annals of statistics* **25**(5), 1189–1232.

Ho, T. K. (1995), Random decision forests, *in* 'Proceedings of 3rd international conference on document analysis and recognition', Vol. 1, IEEE, pp. 278–282.

Nijman, S., Leeuwenberg, A., Beekers, I., Verkouter, I., Jacobs, J., Bots, M., Asselbergs, F., Moons, K. & Debray, T. (2022), 'Missing data is poorly handled and reported in prediction model studies using machine learning: a literature review', *Journal of clinical epidemiology* **142**, 218–229.

Quinlan, J. (1993), *C4.5: Programs for Machine Learning*, Morgan Kaufmann series in machine learning, Elsevier Science.

Twala, B. E., Jones, M. & Hand, D. J. (2008), 'Good methods for coping with missing data in decision trees', *Pattern Recognition Letters* **29**(7), 950–956.

Appendix A. Tree algorithms

Algorithm 2 Missing In Attributes training algorithm

Let

- $(y_i, x_i)_{i\in\mathcal{I}}$, where $y_i \in \mathcal{Y}$, $x_{ij} \in \mathcal{X}_j$, $j = 1, \dots, p$, be the training data
- $\mathcal{L}((y_i)_{i\in\mathcal{I}}, \delta)$ be the loss given parameter δ
- $\delta_{\mathcal{I}}$ be the minimizing parameter of $\mathcal{L}((y_i)_{i\in\mathcal{I}}, \delta)$
- $\mathcal{I}_{jl} = \{i \in \mathcal{I} : x_{ij} \in \mathcal{X}_j^l\}$, $\mathcal{I}_{jr} = \{i \in \mathcal{I} : x_{ij} \in \mathcal{X}_j^r\}$
- $d_{\max}$ be the maximum depth
- n be the minimum number of samples per node

Define training function $\mathcal{T}\left((y_i, x_i)_{i\in\mathcal{I}}, d\right) \to h(x)$:

If $d = d_{\max}$ or $|\mathcal{I}| = n$:

Output

$$h(x) = \delta_{\mathcal{I}}$$

Else:

Fit

Find j, $\mathcal{X}_j^l$, and $\mathcal{X}_j^r$ that minimize

$$\mathcal{L}\left((y_i)_{i\in\mathcal{I}_{jl}}, \delta_{\mathcal{I}_{jl}}\right) + \mathcal{L}\left((y_i)_{i\in\mathcal{I}_{jr}}, \delta_{\mathcal{I}_{jr}}\right)$$

such that $|\mathcal{I}_{jl}| \geq n$, $|\mathcal{I}_{jr}| \geq n$, $\mathcal{X}_j^l \cup \mathcal{X}_j^r = \mathcal{X}_j \cup \{\text{nan}\}$

Grow

$$h_l(x) = \mathcal{T}\left((y_i, x_i)_{\mathcal{I}_{jl}}, d+1\right)$$

$$h_r(x) = \mathcal{T}\left((y_i, x_i)_{\mathcal{I}_{jr}}, d+1\right)$$

Output

$$h(x) = \begin{cases} h_l(x), & x_{\cdot j} \in \mathcal{X}_j^l \\ h_r(x), & x_{\cdot j} \in \mathcal{X}_j^r \end{cases}$$

Algorithm 3 Majority Rule training algorithm

Let

- $(y_i, x_i)_{i\in\mathcal{I}}$, where $y_i \in \mathcal{Y}$, $x_{ij} \in \mathcal{X}_j$, $j = 1, \dots, p$, be the training data
- $\mathcal{L}((y_i)_{i\in\mathcal{I}}, \delta)$ be the loss given parameter δ
- $\delta_{\mathcal{I}}$ be the minimizing parameter of $\mathcal{L}((y_i)_{i\in\mathcal{I}}, \delta)$
- $\mathcal{I}_{jl} = \{i \in \mathcal{I} : x_{ij} \in \mathcal{X}_j^l\}$, $\mathcal{I}_{jr} = \{i \in \mathcal{I} : x_{ij} \in \mathcal{X}_j^r\}$, and $\mathcal{I}_{jm} = \{i : x_{ij} \text{ missing}\}$
- $d_{\max}$ be the maximum depth
- n be the minimum number of samples per node

Define training function $\mathcal{T}\left((y_i, x_i)_{i\in\mathcal{I}}, d\right) \to h(x)$:

If $d = d_{\max}$ or $|\mathcal{I}| = n$:

Output

$$h(x) = \delta_{\mathcal{I}}$$

Else:

Fit

Find j, $\mathcal{X}_j^l$, and $\mathcal{X}_j^r$ that minimize

$$\mathcal{L}\left((y_i)_{i\in\mathcal{I}_{jl}}, \delta_{\widetilde{\mathcal{I}}_l}\right) + \mathcal{L}\left((y_i)_{i\in\mathcal{I}_{jr}}, \delta_{\widetilde{\mathcal{I}}_r}\right)$$

where

$$\widetilde{\mathcal{I}}_l = \begin{cases} \mathcal{I}_{jl}, & |\mathcal{I}_{jl}| \le |\mathcal{I}_{jr}| \\ \mathcal{I}_{jl} \cup \mathcal{I}_{jm}, & |\mathcal{I}_{jl}| > |\mathcal{I}_{jr}| \end{cases}$$

and vice versa for $\widetilde{\mathcal{I}}_r$, such that $\left|\widetilde{\mathcal{I}}_l\right| \ge n$, $\left|\widetilde{\mathcal{I}}_r\right| \ge n$, $\mathcal{X}_j^l \cup \mathcal{X}_j^r = \mathcal{X}_j$

Grow

$$h_l(x) = \mathcal{T}\left((y_i, x_i)_{\widetilde{\mathcal{I}}_l}, d+1\right)$$

$$h_r(x) = \mathcal{T}\left((y_i, x_i)_{\widetilde{\mathcal{I}}_r}, d+1\right)$$

Output

$$h(x) = \begin{cases} h_l(x), & x_{\cdot j} \in \mathcal{X}_j^l \text{ or } x_{\cdot j} \text{ missing and } |\mathcal{I}_{jl}| \ge |\mathcal{I}_{jr}| \\ h_r(x), & x_{\cdot j} \in \mathcal{X}_j^r \text{ or } x_{\cdot j} \text{ missing and } |\mathcal{I}_{jl}| < |\mathcal{I}_{jr}| \end{cases}$$

Algorithm 4 Fractional Case training algorithm

Let

- $(y_i, x_i, w_i)_{i\in\mathcal{I}}$, where $y_i \in \mathcal{Y}$, $x_{ij} \in \mathcal{X}_j$, $j = 1, \dots, p$, $w_i \in [0, 1]$ be the training data
- $\mathcal{L}((y_i, w_i)_{i\in\mathcal{I}}, \delta)$ be the loss given parameter δ
- $\delta_{\mathcal{I}}$ be the minimizing parameter of $\mathcal{L}((y_i)_{i\in\mathcal{I}}, \delta)$
- $\mathcal{I}_{jm} = \{i \in \mathcal{I} : x_{ij} \text{ missing}\}$, $\mathcal{I}_{jl} = \{i \in \mathcal{I} : x_{ij} \in \mathcal{X}_j^l\} \cup \mathcal{I}_{jm}$, $\mathcal{I}_{jr} = \{i : x_{ij} \in \mathcal{X}_j^r\} \cup \mathcal{I}_{jm}$
- $d_{\max}$ be the maximum depth
- n be the minimum total sample weight per node

Define training function $\mathcal{T}\left((y_i, x_i)_{i\in\mathcal{I}}, d\right) \to h(x)$:

If $d = d_{\max}$ or $|\mathcal{I}| \le n$:

Output

$$h(x) = \delta_{\mathcal{I}}$$

Else:

Fit

Find j, $\mathcal{X}_j^l$, and $\mathcal{X}_j^r$ that minimize

$$\mathcal{L}\left((y_i, w_i^l)_{i\in\mathcal{I}_{jl}}, \delta_{\mathcal{I}_{jl}}\right) + \mathcal{L}\left((y_i, w_i^r)_{i\in\mathcal{I}_{jr}}, \delta_{\mathcal{I}_{jr}}\right)$$

where

$$w_i^l = \begin{cases} w_i, & x_{ij} \in \mathcal{X}_j^l \\ w_i \frac{|\mathcal{I}_{jl} \setminus \mathcal{I}_{jm}|}{|\mathcal{I} \setminus \mathcal{I}_{jm}|}, & x_{ij} \in \{\texttt{nan}\} \end{cases}$$

such that $\sum_{i\in\mathcal{I}_{jl}} w_i^l \ge n$, and vice versa for r, and $\mathcal{X}_j^l \cup \mathcal{X}_j^r = \mathcal{X}_j$

Grow

$$h_l(x) = \mathcal{T}\left(\left(y_i, x_i, w_i^l\right)_{\mathcal{I}_{jl}}, d+1\right)$$

$$h_r(x) = \mathcal{T}\left(\left(y_i, x_i, w_i^r\right)_{\mathcal{I}_{jr}}, d+1\right)$$

Output

$$h(x) = \begin{cases} h_l(x), & x_{\cdot j} \in \mathcal{X}_j^l \\ h_r(x), & x_{\cdot j} \in \mathcal{X}_j^r \\ w_l h_l(x) + w_r h_r(x), & x_{\cdot j} \text{ missing} \end{cases}$$