The Journal of Supercomputing (2026) 82:359
https://doi.org/10.1007/s11227-026-08503-8

Sodet—synthetic oversampling decision trees

Joaquim Fernando Pinto da Costa [1] · **Hugo Alonso** [2,3]

Received: 3 October 2025 / Accepted: 3 April 2026

Abstract
In this work, we present a novel methodology for decision trees that use oversampling, not before tree construction (in the entire dataset), but inside each internal node (and corresponding input space region) of the tree. This strategy proves to be successful in fighting the greedy nature of decision trees. We take also into consideration the nature of the input variables, not just quantitative or binary, and also introduce the use of novel distances between instances that can also be used in other contexts. The application of our methodology to a significant number of datasets, thirteen, both balanced and imbalanced problems, shows the relevance of our approach when compared to CART and C5.0. Although our experiments were conducted on a standard computing platform, the proposed approach is well suited for high-performance computing environments, since node-level oversampling and distance computations can be efficiently parallelized, enabling the method to scale to large and high-dimensional datasets.

Keywords Decision trees · Oversampling · Synthetic samples · Data augmentation · Rare classes

1 Introduction

This work is devoted to the introduction of a novel methodology to build decision trees. It is well known that decision trees are nowadays very common in almost all of the application domains, but unfortunately a single tree is rarely enough because the error rates are usually not competitive with other state-of-the-art models. For that reason, researchers tend to use trees in ensemble methods, like XGBoost or random forests,

✉ Joaquim Fernando Pinto da Costa
jpcosta@fc.up.pt

Hugo Alonso
hugo.alonso@upt.pt

[1] CMUP and Departamento de Matemática, Faculdade de Ciências, Universidade do Porto, Rua do Campo Alegre, s/n, 4169-007 Porto, Portugal

[2] REMIT and Universidade Portucalense, Rua Dr. António Bernardino de Almeida, 541, 4200-072 Porto, Portugal

[3] CIDMA, Universidade de Aveiro, Campus Universitário de Santiago, s/n, 3810-193 Aveiro, Portugal

Published online: 19 April 2026

which show a much lower error rate. Unfortunately, by using linear combinations of hundreds or thousands of trees, we lose much of the greatest property of a single tree, which is its interpretability. Because of that, it is possible for instance to summarize the ensemble with a single tree, called distilled tree, which is built to predict the output of the ensemble, rather than the class. This shows that researchers continue to be interested in making improvements in decision trees alone.

Our aim here is also to have a single tree, built from the training set directly, with a better performance. The main idea consists in fighting the greedy nature of trees, which is, in our opinion, the main reason for the underperformance of this model. It is well known that during construction, trees run out of data very quickly. For instance, the two descendant nodes of the root have only, on average, half of the initial observations, and so on. We fight that greedy nature by imputing more observations, synthetic observations, inside each internal node of the tree. These synthetic observations are created using the sample of data corresponding to that node, and not the entire dataset.

Having stated our main idea, now many possibilities opened to us. We can decide how many synthetic observations to create in each node of the tree; we can profit from the creation of synthetic observations to force, or not, the descendant nodes to have a similar number of observations; we can force, or not, that inside each node the different classes are more or less balanced as well. All of this will be considered in our work and will be achieved by using oversampling, in line with what happens in SMOTE [5]. However, we do not oversample the entire dataset, before tree construction. What we do is to oversample inside only the region of the input space corresponding to each internal node.

There has been extensive work on oversampling, after SMOTE [5], mainly on the problem of rare classes; for example, see [1, 4, 10, 13, 17]. In fact, oversampling can also be used to address the problems of bias in data, which is common in medical data for instance, where some socio-economic groups are under-represented. However, the way we use oversampling here is totally different, as will be seen.

In addition, given that with oversampling we need to find the nearest neighbors of a certain instance, the use of an appropriate distance function is crucial. We describe the distances that we use, and in particular for nominal variables, where we use the chi-square distance of multiple correspondence analysis, and for ordinal variables, where we introduce a novel distance. The way we find the nearest neighbors is also new. Apart from that, the way we create synthetic observations for qualitative data is new, and not just the mode as in SMOTE NC.

We apply our methodology to thirteen datasets, where we show the competitiveness of our results when compared to the CART usual trees [3] and also C5.0 [19]; both with and without oversampling.

2 Review of methods combining resampling with decision trees

We have not seen any method which can compare to ours regarding the way we use oversampling. Most of the methods deal with resampling data in order to have a more balanced dataset and then apply a classification algorithm, decision trees or other. We have not come across any methodology doing oversampling inside the nodes of a

decision tree. Nevertheless, we will present here some references which use resampling together with decision trees.

The seminal paper of SMOTE (Synthetic Minority Oversampling Technique) [5], widely used to address class imbalance, has significant applications in decision trees. In [11], various techniques for handling imbalanced datasets, including both oversampling and undersampling methods, are reviewed. Their focus is to provide a critical review of the nature of the problem, the state-of-the-art technologies, and the current assessment metrics used to evaluate learning performance under the imbalanced learning scenario. In [6], insights into the challenges and methods for dealing with imbalanced datasets are provided, with a specific focus on decision trees and the use of oversampling and undersampling techniques. Additionally, the distribution of the testing data may differ from that of the training data, and the true misclassification costs may be unknown at learning time. The author discusses some of the sampling techniques used for balancing the datasets, and the performance measures more appropriate for mining imbalanced datasets. In [21], three resampling approaches based on class instance density across feature value intervals are proposed to address imbalanced data learning: an oversampling approach, an undersampling approach, and a hybrid approach that combines oversampling with undersampling. The authors consider several classification algorithms in their experiments, including decision trees, and conclude that the best results are achieved with the hybrid approach when applied with decision trees. Nevertheless, they recognize that the oversampling component of the method needs improvement.

3 SODET—synthetic oversampling decision trees

In a certain node t of a decision tree, containing n_t observations, which is divided into two descendant nodes, t_L and t_R, with, respectively, n_{t_L} and n_{t_R} observations, let $p_L = \frac{n_{t_L}}{n_t}$ and $p_R = \frac{n_{t_R}}{n_t}$ be the proportions of observations which go, respectively, to the left and to the right.

In the usual decision trees, the descendants of node t, t_L and t_R, have, on average, 50% of the observations. That is why trees are considered to be a very greedy algorithm, making them run out of data very quickly. For that reason, although trees are a very interpretable algorithm, they provide usually a very simple decision rule, with a somewhat small accuracy.

Our aim is thus to introduce synthetic oversampling new data, in line with what happens in SMOTE [5], in the descendants of each node. In order to be clear, we do not generate artificial data in the entire dataset, as many methods do. Our synthetic data will be generated, step by step, in each node of the decision tree (apart from the root), thus affecting only the corresponding region of the input space, rather than the entire data space. For that, we must start by deciding how many observations, on average, we want to have in each descendant, instead of the usual 50%. Let us introduce a hyperparameter here, β, so that we will have, on average, $100(1-\beta)\%$ in each descendants of node t, t_L and t_R. For instance, if we want that each descendant has, on average, $95\% n_t$ observations, we must choose $\beta = 0.05$.

The number of observations in each descendant, $p_L n_t$ and $p_R n_t$, will be multiplied by a constant C, in order to get the desired average of $100(1-\beta)\%$,

$$\frac{C p_L n_t + C(1-p_L)n_t}{2} = (1-\beta)n_t, \tag{1}$$

that is, $C = 2(1-\beta)$. Thus, the number of observations in t_L will go from n_{t_L} to $2(1-\beta)p_L n_t$ and the number of observations in t_R will go from n_{t_R} to $2(1-\beta)p_R n_t$. We will therefore have to create $2(1-\beta)p_L n_t - p_L n_t = (1-2\beta)n_{t_L}$ synthetic observations in the left node and $2(1-\beta)p_R n_t - p_R n_t = (1-2\beta)n_{t_R}$ synthetic observations in the right node.

3.1 Getting more balanced descendants

If the two descendants, t_L and t_R, have very imbalanced proportions, that is, if the two proportions p_L and p_R are quite different, we might want to profit from the generation of synthetic data in order to get a more balanced division. In that case we would create a larger proportion of synthetic observations in the least frequent descendant and a smaller proportion in the other. Let $p^t_{min} = min(p_L, p_R)$. In order to do that, we introduce here another hyperparameter, $\gamma \geq 1$; now, the number of observations in the least frequent descendant will be multiplied by $\gamma C = 2(1-\beta)\gamma$, instead of C. As we want that in the end, the two descendants have, on average, the same proportion of observations as before, that is, $100(1-\beta)\%n_t$, the number of observations in the other descendant (the more frequent one) will have to be multiplied by $2(1-\beta)\frac{1-\gamma p^t_{min}}{1-p^t_{min}}$. Suppose, for instance, that the least frequent descendant is t_L. That is, in the least frequent descendant we will end up with $2(1-\beta)\gamma p_L n_t$ observations and thus we will have to create

$$ns_{t_L} = (2(1-\beta)\gamma - 1)n_{t_L}, \quad \text{if } \mathrm{p_L} < \mathrm{p_R} \tag{2}$$

synthetic observations. In the other descendant, t_R, we will end up with $2(1-\beta)\frac{1-\gamma p_L}{1-p_L}(1-p_L)n_t$ observations and so we would have to create

$$ns_{t_R} = (2(1-\beta)\frac{1-\gamma p^t_{min}}{1-p^t_{min}} - 1)n_{t_R}, \quad \text{if } \mathrm{p_L} < \mathrm{p_R} \tag{3}$$

synthetic observations. In case it is t_R the least frequent, we swap t_L with t_R in these expressions.

For example, if our node t has 100 observations and 30 of them go left and 70 go right, and if we want to have, on average, each descendant with 95% of the observations in t, we have to use $\beta = 0.05$; if in addition, we do not want to change the balance of the two descendants, that is $\gamma = 1$, then we will have to create $(2(1-\beta)\gamma - 1)30 = 27$ synthetic observations in the least frequent node, the left node, to join the other 30. For the other node, we would have to create $(2(1-\beta)\frac{1-\gamma p^t_{min}}{1-p^t_{min}} - 1)70 = 63$ synthetic

observations to join the other 70. We would end up with 57 observations in the left node and 133 in the right one. On average we would have then 95 observations in each descendant and the proportions would be the same (30 and 70%). If we want to have more balanced descendants and we choose, for instance, $\gamma = 1.3333$, then, we would have to create 46 synthetic observations in the least frequent descendant, the left one, and 44 in the other. We would end up with 76 observations in the left node and 114 in the right one, having thus the same average of 95% of the observations per descendant but this time, the left node would end up with 40% of the observations and the right node, 60%. If we want to go further and force the two descendants to have equal proportions, we must choose $\gamma = 1.6667$ (see below); then, we would have to create 65 synthetic observations in the left node and 25 in the right one, ending up with 95 observations in each.

3.2 Range of values for β and γ

Although in principle very general values for β and γ could be considered, we will limit ourselves here to the values that we consider more appropriate. For instance, when $\beta = 0.5$, we are in the original situation where the descendants of a node, t_L and t_R, have on average 50% of the observations of the parent node t and so, there is no need for synthetic data creation. We could, in theory, consider values for β greater than 0.5, but that would mean that we would have to do undersampling and that is not the purpose of this work. As for the minimum value for β, any value greater than zero is acceptable, in principle, although in practice, that has to be estimated and each particular problem will have a specific value of β. Again, in theory, we could consider values smaller than or equal to zero. In summary, in this work, the range of values for β will be

$$0 < \beta \leq 0.5 \tag{4}$$

In order to avoid extremely large trees, which can happen especially when β is near zero, we will not maintain constant the value of β during the construction of the tree. We start with a value β_1 in the first level of the tree that satisfies Eq. (4). That is, for the two descendants of the root of the tree, we will use that value β_1, which has to be chosen, for instance, with cross-validation. Then, we will successively linearly increase the value of β until it reaches 0.5 in the level L of the tree. From this level on, there is no synthetic data creation. Thus, our final range for the values of β is

$$\beta(l) = \frac{\beta_1(L-l) + 0.5(l-1)}{L-1}, \quad 2 \leq l \leq L \tag{5}$$

Again, the value of the level L where the synthetic data creation stops (that is, when $\beta(l) = 0.5$), can be decided beforehand or estimated using, for instance, cross-validation. In our experiments we have tried large values for L, for instance 20, but the best results were obtained for smaller values and so, we will only consider here values for L between 2 and 5.

The situation for the range of values for γ is somewhat similar. When $\gamma = 1$ we are in the situation where the descendants of a node, t_L and t_R, will maintain the same initial proportions, p_L and p_R, respectively. We could in principle consider values for γ smaller than 1, meaning that the gap between the two proportions, p_L and p_R, would increase and therefore, the descendants would be less balanced than they were originally. As we have seen above, in the least frequent descendant (for instance t_L) we will have to create $(2(1-\beta)\gamma - 1)n_{t_L}$ synthetic observations. That means that $(2(1-\beta)\gamma - 1)$ has to be greater than or equal 0 (otherwise we would have to do undersampling), so $\gamma \geq \frac{1}{2(1-\beta)}$.

As for the upper limit, in this work we will consider the value of γ that makes the two proportions, p_L and p_R, equal. We do not consider here the possibility that the least frequent descendant, for instance t_L, becomes, in the end, more dominant than the other, t_R, although, in theory, that could be considered. Our limit in this work will be thus the value of γ that makes the two proportions equal, in case that is possible with oversampling without changing the global mean value of $100(1-\beta)\%$ in each node; that is, assuming t_L is the least frequent descendant (see above),

$$2(1-\beta)\gamma\, p_L n_t \leq 2(1-\beta)\frac{1-\gamma\, p_L}{1-p_L}(1-p_L)n_t, \tag{6}$$

which means that $\gamma \leq \frac{1}{2p_L}$. On the other hand, because in t_R, we will have to create $(2(1-\beta)\frac{1-\gamma\, p^t_{min}}{1-p^t_{min}} - 1)n_{t_R}$ synthetic observations, that means that $(2(1-\beta)\frac{1-\gamma\, p^t_{min}}{1-p^t_{min}} - 1)$ has to be greater than or equal 0. In summary, the range of values that we will use in this work for γ are

$$MAX\left(1, \frac{1}{2(1-\beta)}\right) \leq \gamma \leq MIN\left(\frac{1}{2p^t_{min}}, \frac{2(1-\beta)-1+p^t_{min}}{2(1-\beta)p^t_{min}}\right) \tag{7}$$

We note that when the value of β is 0.5, that is, there is no synthetic data creation, the value of γ is 1, which means that the balance between the two descendants of a node t remains as it is. This is to be expected since when there is no synthetic observations, we get the usual decision trees.

In (7) we have the possible range of values for γ with oversampling. For other values, we would have to do undersampling, which is not our aim here. We see that the range of values for γ depend on the nodes and we could consider a grid of values, in each node. For instance, if we denote by $R_{\gamma,t}$ the range of values for γ,

$$R_{\gamma,t} = MIN\left(\frac{1}{2p^t_{min}}, \frac{2(1-\beta)-1+p^t_{min}}{2(1-\beta)p^t_{min}}\right) - MAX\left(1, \frac{1}{2(1-\beta)}\right), \tag{8}$$

we could use the values

$$\begin{aligned} &\gamma_1 = MAX\Big(1, \frac{1}{2(1-\beta)}\Big); \quad \gamma_2 = \gamma_1 + 25\%R_{\gamma,t}; \quad \gamma_3 = \gamma_1 + 50\%R_{\gamma,t}; \\ &\gamma_4 = \gamma_1 + 75\%R_{\gamma,t}; \quad \gamma_5 = \gamma_1 + R_{\gamma,t}, \end{aligned} \tag{9}$$

or a finer grid. We can also use, for instance, the minimum, the maximum and the mean values of that range. That is what we did above in that small example, when we considered the values for $\gamma = 1, 1.3333, 1.6667$. In practice, however, the user must decide which value of γ to use but this value depends on the node, it is not a constant during the construction of the tree; for each node it varies in a certain interval as seen above. For that reason, the parameter that the user will choose is Q_γ, the quantile of that interval above (8). For instance, if the user wants to choose always the smallest possible value of that interval above, that is, the user does not want to balance the two descendants t_L and t_R, then the value $Q_\gamma = 0$ should be chosen. If the user wants to choose always the largest possible value, that is, the maximum of each interval, which means that each descendant node will have the same number of observations, then the value $Q_\gamma = 1$ should be chosen. For other intermediate values, the user should choose a real number in [0, 1]. For instance, if the user chooses $Q_\gamma = 0.75$, it means that in each node, $\gamma = MAX(1, \frac{1}{2(1-\beta)}) + 75\%R_{\gamma,t}$. In practice, however, the user might not know what is the best value for this parameter γ and that can be estimated, for instance, using cross-validation. The same applies to the parameter β above which can be estimated in conjunction with γ, or separately, using cross-validation. For instance, we might decide on one of them and estimate the other using cross-validation, or estimate both at the same time with cross-validation as well.

3.3 Distribution of synthetic data per class

For each value of β and γ, in the left node t_L, we have to create ns_{t_L} synthetic observations and, in the right node, ns_{t_R} synthetic observations [see Eqs. (2) and (3) above]. We will now consider different strategies concerning how many synthetic observations need to be created for each class.

Firstly, if the problem is not a binary classification task in which one of the classes is rare, we will proceed according to one of the following two approaches:

(1) Unchanged classes (UC) approach: we will maintain the proportions per class that already exist in the node, that is, for class $\omega_k, k = 1, 2, \ldots, K$, we will create $ns_{t_L} p_{t_L,k}$ observations, where $p_{t_L,k}$ is the proportion of original observations from class ω_k and node t that reached node t_L; similarly for the node t_R, where we will create $ns_{t_R} p_{t_R,k}$ observations, where $p_{t_R,k}$ is the proportion of original observations from class ω_k and node t that reached node t_R.
(2) Balanced classes (BC) approach: if we want to profit from the generation of synthetic data and try to make the proportions per class constant inside each node, we have to do the following:
If $p_L < p_R$ we have to generate for class $\omega_k, k = 1, 2, \ldots, K$, of node t_L, $ns_{t_L,k}$ synthetic observations and $ns_{t_R,k}$ observations in the case of node t_R, where

$$ns_{t_L,k} = \frac{2(1-\beta)\gamma p_L n_t}{K} - n_{t_L,k} \quad \text{and} \quad ns_{t_R,k} = \frac{2(1-\beta)(1-\gamma p_L) n_t}{K} - n_{t_R,k} \tag{10}$$

$n_{t_L,k}$ and $n_{t_R,k}$ are the number of observations inside class ω_k in each node before data creation. In case it is t_R the least frequent, we swap t_L with t_R in these expressions.

If any of the values $ns_{t_L,k}$ or $ns_{t_R,k}$ is negative, we do not generate values for class ω_k in that node, which means that we cannot completely force the proportions per class to be exactly the same in the node.

Secondly, if the problem is a binary classification task in which one of the classes—say, class ω_k—is rare, then we will proceed according to the following approach:

(3) Rare class (RC) approach for binary classification: We recall that the value of β serves to control the amount of oversampling and the value of γ serves to balance the distribution of the two descendants. None of these two parameters changes the class proportions inside each node. In the previous approach, we saw how to balance the classes inside each node for a non binary classification task. For a binary classification task, we will now introduce a third parameter μ for that purpose. In node t_L, we will create $ns_{t_L}\mu p_{t_L,k}$ observations in class ω_k (the rare class), and $ns_{t_L} - ns_{t_L}\mu p_{t_L,k}$ in the other class, where μ is defined as:

$$\mu = \begin{cases} 1 & \text{if} \quad p_{t_L,k} > 20\% \\ 2 & \text{if} \quad 10\% < p_{t_L,k} \leq 20\% \\ 3 & \text{if} \quad p_{t_L,k} \leq 10\% \end{cases} \tag{11}$$

That is, we duplicate or triplicate the frequency of a rare class inside the node. Thus, the lower the proportion $p_{t_L,k}$ of original observations from class ω_k and node t that reached node t_L—i.e., the greater the imbalance between the two classes in node t_L—the greater the number of observations we will create in the rarer class ω_k. Note that taking $\mu = 1$ corresponds to maintaining the class proportions already present in the node. We proceed similarly in node t_R.

4 Distance used

In real data, we can get different types of scales in our variables: qualitative nominal, qualitative ordinal and quantitative. Quantitative variables can be further divided into discrete and continuous (ratio or interval) scale. It is often the case that researchers just use a distance for numerical variables (quantitative continuous) or one that combines numerical with nominal. In this work, we will take into account the four scales.

4.1 Distance between objects described by qualitative nominal variables

In the case of a nominal variable X_j, researchers use a number of different distances between individuals (lines) i and l. For instance,

$$d^j(x_{ij}, x_{lj}) = \begin{cases} 0 & \text{if } x_{ij} = x_{lj} \\ 1 & \text{if } x_{ij} \neq x_{lj}. \end{cases} \tag{12}$$

This is the simplest distance that can be used. When we want to combine in the end the different distances between the four types of data into a single distance, this one might not be appropriate and for that reason, there is also the distance used in SMOTE NC [5]:

$$d^j(x_{ij}, x_{lj}) = \begin{cases} 0 & \text{if } x_{ij} = x_{lj} \\ \text{Med} & \text{if } x_{ij} \neq x_{lj} \end{cases} \tag{13}$$

where Med represents the median value of the standard deviations of the continuous variables in the dataset. Another possible distance is the modified version of the Value Difference Metric [7], which was originally introduced to measure the distance between instances in a classification problem with nominal features,

$$d^j(x_{ij}, x_{lj}) = \sum_{k=1}^{K} \left| \frac{n_{ik}}{n_1} - \frac{n_{jk}}{n_2} \right|^{\eta} \tag{14}$$

where K represents the number of classes in the classification problem, n_{ik} and n_{jk} are the number of occurrences of categories x_{ij} and x_{lj} in class ω_k and n_1, n_2 are the total number of occurrences of those modalities; η is a constant. Naturally, this distance is only valid for classification problems and it is difficult to know its maximum value for a specific situation.

In this work, the distance that we will use for a qualitative nominal variable is the distance used in multiple correspondence analysis [8], the chi-square distance, which takes the number of categories, q_j, into consideration, is not only for classification problems and takes $C_2^{q_j} + 1$ possible values and not just 2 (unless $q_j = 2$, the minimum possible value). If $q_j = 1$, that is, X_j is a constant, the distance is zero. It is thus a richer measure of distance between two objects described by a nominal variable. To calculate this distance, we must first start to codify each nominal variable with q_j categories, into q_j binary variables, as it is done in multiple correspondence analysis. In order to simplify notation, let us call these binary variables $Z_1, Z_2, ..., Z_{q_j}$. Then, the distance is

$$d^j_{\chi^2}(x_{ij}, x_{lj}) = \sum_{m=1}^{q_j} \frac{1}{z_{.m}} (z_{im} - z_{lm})^2, \tag{15}$$

where $z_{im} = 1$ if individual i takes category m of that categorical variable and 0 otherwise; similarly for z_{lm}. As for $z_{.m}$, it is the total number of individuals in the training set that share that category, m, of that categorical variable. It is thus clear that this distance takes $C_2^{q_j} + 1$ possible values, starting with zero and that the maximum value for this distance is

$$M_j = \frac{1}{\min(z_{.m})} + \frac{1}{\min_2(z_{.m})}, \tag{16}$$

where $\min_2(z_{.m})$ is the second smallest value of $z_{.1}, z_{.2}, ..., z_{.q_j}$. Knowing this maximum value allows us to normalize this distance, for instance dividing by the maximum value, so that we can combine it with the other distances into a final distance as we will see below. It is also clear that this distance gives higher weights to the categories that are less frequent in the training set, because it divides each binary contribution by the total number of individuals in the training set which share that category, $z_{.m}$. This will also be a useful property in what follows, as we will see.

4.2 Distance between objects described by qualitative ordinal variables

Ordinal variables are somewhere in between nominal and quantitative variables and for that reason, most often they are treated as nominal, ignoring therefore the natural order between the categories, or codified into a numerical scale. For instance, in the original formulation of the Gower's distance [9], nominal and ordinal variables are not distinguished.

Kaufman and Rousseeuw [16] suggested to do the following. If X_j is ordinal taking q_j different or distinct values, namely $v_1 < \ldots < v_{q_j}$, we assign the integer numbers 1, 2, ...,q_j to these categories and codify X_j into a new variable Z_j such that

$$z_{ij} = \frac{\text{ord}\left(x_{ij}\right) - 1}{q_j - 1}, \tag{17}$$

where ord represents the integer corresponding to the category of x_{ij}. After that, the variable is treated as numerical.

Podani [18] suggested also a very similar treatment for ordinal variables but using ranks. Consider a new variable R_j whose value for the i-th instance corresponds to the rank of the value of X_j for that instance determined over all n instances, i.e., r_{ij} corresponds to the rank of x_{ij} in the list of values $x_{1j}, \ldots, x_{nj}$ present in the j-th column of the input matrix $\boldsymbol{X}_{n\times m}$ of the training data. An average rank is computed in case of ties. Podani suggests taking

$$d^j(x_{ij}, x_{lj}) = \begin{cases} 0 & \text{if } r_{ij} = r_{lj} \\ \frac{|r_{ij} - r_{lj}| - (T_{ij}-1)/2 - (T_{lj}-1)/2}{\max\{r_{1j},\ldots,r_{nj}\} - \min\{r_{1j},\ldots,r_{nj}\} - (T_{j,\max}-1)/2 - (T_{j,\min}-1)/2} & \text{if } r_{ij} \neq r_{lj} \end{cases} \tag{18}$$

where T_{ij} is the number of instances that have the same rank of instance i in variable j (including instance i), T_{lj} is similar to T_{ij}, but for instance l, $T_{j,\max}$ is the number of instances that have maximum rank in variable j and $T_{j,\min}$ is similar to $T_{j,\max}$, but for the case of minimum rank.

Jajuga et al. [14] proposed a general distance measure, based on a generalized correlation coefficient, which treats similarly ordinal and continuous variables. All of the ordinal variables are treated simultaneously and the distance between the two vectors of ordinal variables corresponding to instances i and l are based on a form of correlation. See [14] for more details.

Given that, as we said in the beginning, ordinal variables are in between nominal and quantitative, we propose here a distance measure for ordinal variables which is a combination of the two. More specifically, our proposal is

$$d^j_{\chi^2_o}(x_{ij}, x_{lj}) = |\text{ord}(x_{ij}) - \text{ord}(x_{lj})| \sum_{m=1}^{q_j} \frac{1}{z_{.m}}(z_{im} - z_{lm})^2, \tag{19}$$

where, as above, ord represents the integer corresponding to a category. See Eq. (15) for further details. In order to normalize this distance and force it to take values in [0, 1], we have to find its maximum in the training set, which is

$$M_{j,o} = \max_{\substack{m=1,\ldots,q_j-1 \\ s=m+1,\ldots,q_j}} (s-m)\left(\frac{1}{z_{.m}} + \frac{1}{z_{.s}}\right) \tag{20}$$

4.3 Distance between objects described by quantitative variables

In the case of quantitative variables, whether measured in discrete, continuous (ratio or interval scales), we will use the Gower's distance [9],

$$d^j_G(x_{ij}, x_{lj}) = \frac{\left|x_{ij} - x_{lj}\right|}{\max\left\{x_{1j}, \ldots, x_{nj}\right\} - \min\left\{x_{1j}, \ldots, x_{nj}\right\}}, \tag{21}$$

which also takes values in the interval [0, 1], making it easy to enter a general final distance (see below), and is popular in data analysis.

4.4 Final distance combining nominal, ordinal and quantitative variables

In order to combine now all of the singular distances seen above—for nominal, ordinal and quantitative variables—into a final distance, we propose the following formula to find the distance between the individuals (lines) i and l of the dataset, in a general data analysis context:

$$\begin{aligned} d_U(x_i, x_l) = &\sum_{j=1}^{p_n} \frac{1}{M_j} d^j_{\chi^2}(x_{ij}, x_{lj}) + \sum_{j=p_n+1}^{p_n+p_o} \frac{1}{M_{j,o}} d^j_{\chi^2_o}(x_{ij}, x_{lj}) \\ &+ \sum_{j=p_n+p_o+1}^{p_n+p_o+p_q} d^j_G(x_{ij}, x_{lj}) \end{aligned} \tag{22}$$

where the expressions of $d^j_{\chi^2}$, $d^j_{\chi^2_o}$ and d^j_G are given by Eqs. (15), (19) and (21) above and p_n is the total number of nominal variables, p_o the total number of ordinal variables and p_q the total number of quantitative variables. For the sake of simplicity, we assume that first we have the nominal variables, then the ordinal ones and finally

Table 1 Weighted frequency table of the categories of variable X_j among the K neighbors of $\mathbf{x}$

m_1	m_2	...	m_{q_j}
$\frac{1}{S_z}\sum_{l=1}^{K}\frac{1}{z_{.1}}z_{l1}$	$\frac{1}{S_z}\sum_{l=1}^{K}\frac{1}{z_{.2}}z_{l2}$	...	$\frac{1}{S_z}\sum_{l=1}^{K}\frac{1}{z_{.q_j}}z_{lq_j}$

the quantitative variables, but the user does not have to care with this; just identify the nature of each variable. Each partial distance has been normalized and takes values in the interval [0, 1].

5 Generation of synthetic cases

In the case of a mix of nominal, ordinal and quantitative data, we need to generate synthetic cases, in order to oversample the data that has reached a certain descendant node, t_L or t_R, of a node t of the decision tree. As in SMOTE NC, a new synthetic case $\tilde{\mathbf{x}}$ is generated in a class C by combining an instance $\mathbf{x}$ randomly selected in C with its K nearest neighbors $\mathbf{x}_1, \ldots, \mathbf{x}_K$ in C. Nevertheless, once we have to generate many synthetic observations, a sample in each descendant node, to speed up the computational process, we will limit our search for the K nearest neighbors inside a random sample of, at most, $3K$ observations of the node and class in question.

For the generation of the values for the quantitative (discrete, continuous; ratio or interval scales) components of $\tilde{\mathbf{x}}$, note that if $\mathbf{x}_\ell$ is a random neighbor from the set of K, and α in [0, 1], we can use the following equation,

$$\tilde{x}_j^{\mathrm{q}} = x_j^{\mathrm{q}} + \alpha\left(x_{\ell j}^{\mathrm{q}} - x_j^{\mathrm{q}}\right), \quad j = 1, \ldots, p_q. \tag{23}$$

The quantitative part of the synthetic case $\tilde{\mathbf{x}}$ is thus a convex linear combination of $\mathbf{x}$ and $\mathbf{x}_\ell$, where $\mathbf{x}_\ell$ is one of the nearest neighbors of $\mathbf{x}$ chosen at random. So, as in SMOTE NC, for the generation of the quantitative synthetic data, only one of the K nearest neighbors is used. The difference is that we use a different distance between instances and so our set of neighbors can be different.

For the generation of the values for the qualitative components (nominal or ordinal scales) of $\tilde{\mathbf{x}}$, on the contrary, we will use all of the K neighbors, as in SMOTE NC, but with an important adaptation. In the case of SMOTE NC, for each qualitative variable, the mode of the K neighbors is considered; naturally, it is likely that more than one mode exists and in that case, one of them is chosen at random.

As said above, in our case, a qualitative variable X_j with q_j categories $(m_1, m_2, \ldots, m_{q_j})$ is transformed into q_j binary variables, one for each category. Taking into account that each binary variable Z_m, has a weight $\frac{1}{z_{.m}}$ in the calculation of the chi-square distances $d^j_{\chi^2}$ (15) and $d^j_{\chi^2_o}$ (19) that we use, we will profit from this and use a weighted frequency of those categories among the K neighbors. In Table 1, where we present those weighted frequencies, z_{li} is 1 if neighbor l has category m_i of categorical variable X_j and 0 otherwise and $S_z = \sum_{i=1}^{q_j}\sum_{l=1}^{K}\frac{1}{z_{.i}}z_{li}$.

Table 2 Convex linear combination between frequencies of **x** and of its K neighbors

m_1	m_2	$\ldots m_{q_j}$
$z_{x1} + \alpha\left(\frac{1}{S_z}\sum_{l=1}^{K}\frac{1}{z_{.1}}z_{l1} - z_{x1}\right)$	$z_{x2} + \alpha\left(\frac{1}{S_z}\sum_{l=1}^{K}\frac{1}{z_{.2}}z_{l2} - z_{x2}\right)$	$\ldots z_{xq_j} + \alpha\left(\frac{1}{S_z}\sum_{l=1}^{K}\frac{1}{z_{.q_j}}z_{lq_j} - z_{xq_j}\right)$

Now, we could take the mode of this frequency table to assign to the synthetic case $\tilde{\mathbf{x}}$, corresponding to categorical variable X_j. Contrary to SMOTE NC, it is unlikely that there is more than one mode, because of the weights $\frac{1}{z_{.m}}$ that we are using. Note that again because of these weights $\frac{1}{z_{.m}}$, rare categories are favored. For instance, if in 5 neighbors 3 share the same category m_1 and 2 share another category m_2, SMOTE NC would take category m_1 to assign to the synthetic case. However, if category m_2 is a rare category, the fact that we have 2 out of 5 with that rare category is certainly more significant than 3 out of 5 of a frequent category and in our case, m_2 is more likely to be chosen, depending on the weights $\frac{1}{z_{.m}}$ that we use. We recall that the way we treat categorical variables is the same as it is done in , multiple correspondence analysis [8].

Instead of taking the mode of Table 1, however, we will go further and, as it is done in the quantitative part of the generation of synthetic observations [see Eq. (23)], we want also to take into account, directly, the information of instance **x** that we are at. Again, in the case of SMOTE NC, the instance is not taken into account for the generation of the categorical part of synthetic cases. For that, we will also use here a convex linear combination between the values in Table 1, which are a weighted mean of the K neighbors categories frequency and the frequencies of instance **x**, as it is done in Table 2.

Again, as in the case of the quantitative part, α is a real random number in [0, 1]; the same random number. $z_{xi} = 1$ if instance **x** takes category m_i of the categorical variables in question, X_j, and 0 otherwise. Finally, the category that we choose to represent the value of the categorical variable X_j corresponding to the synthetic case $\tilde{\mathbf{x}}$ is the mode of Table 2. Thus, it is clear that because all but one of the z_{xi} values are zero, we either choose the category of instance **x**, if α is sufficiently close to zero, or the mode of weighted frequencies of its neighbors in Table 1, in case α is sufficiently close to one.

6 Empirical study

In this section, we present an empirical study comparing first SODET trees with CART trees across 11 classification problems, including both binary and multiclass tasks. Of these, seven are balanced and four are imbalanced. Later we run another experiment comparing SODET with CART and C5.0, where we also allow these two methods to use synthetic data. In addition, we include another two datasets; thirteen in total. Section 6.1 describes the datasets used, while Sect. 6.2 outlines the methodology. The results and their discussion are presented in Sect. 6.3.

6.1 Datasets

Overall, we consider the 13 datasets described in Table 3, which are available from the UC Irvine Machine Learning Repository (http://www.archive.ics.uci.edu/) or on Kaggle (https://www.kaggle.com). For each dataset, the maximum imbalance ratio (Max IR) is reported, defined as the ratio between the number of samples in the most frequent class and the number of samples in the least frequent class. The datasets are presented in increasing order of Max IR. In the first nine datasets, the Max IR is less than 4 and, in most cases, close to 1. In the remaining four datasets, the Max IR is close to 10 or higher. Therefore, the first nine datasets are considered balanced, while the remaining four are classified as imbalanced [12].

Initially, the cases in the Body Mass Index dataset were grouped into six classes. We transformed this multiclass classification problem into a binary classification problem, where the goal is to distinguish individuals in the "extremely weak" and "weak" groups from those in the remaining groups. In the Sleep health and lifestyle dataset, the original categories "insomnia" and "apnea" were merged into a single group, denoted as "sleep disorder", thereby defining a binary classification task aimed at distinguishing between individuals with "no sleep disorder" and those with a "sleep disorder".

The Gene Cancer and Credit Card Fraud datasets initially contained more features, but we applied feature selection to reduce the computational burden of our experiments. The Gene Cancer dataset included 20,531 features, and we performed feature selection using one-way analysis of variance (ANOVA) [20]. For each feature, we calculated the F-score with respect to the class variable. A high F-score indicates that the feature is effective at distinguishing between classes, while a low F-score suggests it is not. Ultimately, we selected the 307 features (1.5%) with the highest F-scores. The Credit Card Fraud dataset contained 30 features, 28 of which were derived from a principal component analysis (PCA) [15] performed to transform confidential data. Of these 28 principal components, we retained the most informative, specifically, the first 12. This is the minimum number of components needed to explain more than 70% of the cumulative explained variance.

The Credit Card Fraud dataset initially contained 284,807 samples. However, to reduce the computational burden of our experiments, we retained only 5000 samples: 50 (1%) from the minority class (positive cases) and 4950 (99%) from the majority class (negative cases). The 50 positive cases correspond to the first 50 positive cases in the original file, and the 4950 negative cases correspond to the first 4950 negative cases.

All datasets were randomly divided into training (two-thirds) and test (one-third) sets, while maintaining the original class proportions.

6.2 Methodology

We used the training data to grow SODET and CART trees, stopping the splitting process at a node when it was pure, i.e., when all observations in the node belonged to the same class, or when each feature had the same value across all observations in the node. The split criterion was Gini's diversity index.

Table 3 Datasets considered in the empirical study

Dataset	# Classes	Max IR	# Instances Training	Test	*#Features*	Feature type
Iris	3	1.0	100	50	4	Quant
Vehicle silhouettes	4	1.1	564	282	18	Quant
Banknote	2	1.2	915	457	4	Quant
Rice	2	1.3	2540	1270	7	Quant
Sleep	2	1.4	250	124	12	Nom., ord., quant
Game	2	1.9	639	319	9	Nom
Kidney disease	2	2.7	104	51	24	Nom., ord., quant
Human resources	2	3.2	10, 000	4999	9	Nom., ord., quant
Gene Cancer	5	3.9	534	267	307	Quant
Bank Loan	2	9.4	3334	1666	12	Nom., quant
Body Mass Index	2	12.9	334	166	3	Nom., quant
Stroke	2	22.4	3273	1636	10	Nom., quant
Credit Card Fraud	2	97.1	3334	1666	14	Quant

Max IR stands for maximum imbalance ratio. Nom., ord., and quant. are the abbreviations for nominal, ordinal, and quantitative, respectively

After growing SODET and CART trees, we pruned the trees using the five pruning methods (PM) described below. We denote by SODET-PMi or CART-PMi the subtree obtained by applying the i-th pruning method to the fully grown SODET or CART tree, respectively.

PM1: In the first method, a tree is pruned at a node if pruning does not increase the classification error on the original training data.
PM2: In the second method, a tree is pruned at a node if pruning does not increase the classification error on both the original training data and the synthetic data.
PM3: In the third method, a tree is pruned at a node if the node is pure in the original training data.

Note that SODET trees incorporate synthetic data into the original training set, whereas CART trees do not. As a result, these first three pruning methods tend to produce different subtrees when applied to the same SODET tree, but they always produce the same subtree when applied to the same CART tree. This means that SODET-PM1, SODET-PM2, and SODET-PM3 tend to differ, while CART-PM1, CART-PM2, and CART-PM3 are identical. We refer to this common CART subtree as CART-PM1/2/3. The fourth and fifth pruning methods are the cost-complexity pruning methods described by [3]. Both methods have a complexity parameter, $\alpha \geq 0$ (not to be confused with the value of α used above in the linear combination), which can be estimated using cross-validation. The higher the value of α, the smaller the pruned tree (i.e., the less complex).

PM4: In the fourth method, the estimate of α is the largest value corresponding to the minimum cross-validation error.

PM5: In the fifth method, the estimate of α is the largest value for which the cross-validation error does not exceed the minimum plus one standard deviation.

In our experiments, neither CART, nor SODET hyperparameters (split criterion, stopping rules such as maximum depth, minimum samples per leaf, minimum samples to split, etc.) were analyzed. For both methods, the situation is similar, that is, we did not analyze the influence these parameters could have. It is a fact that SODET has other hyperparameters that are not usual in decision trees and the purpose of our paper is to study the effect of these hyperparameters. As for the stopping rules, all of the five pruning methods were applied to both CART and SODET. Before pruning with any of the five methods, when we grew SODET trees using the balanced datasets in Table 3, we considered two possible distributions of synthetic data per class, as described in Sect. 3.3 for cases where the problem is not a binary classification task in which one of the classes is rare. The first approach maintains the class proportions already present in each node, while the second attempts to equalize the class proportions within each node. Recall that the first approach is referred to as the "Unchanged classes" (UC) approach, and the second as the "Balanced classes" (BC) approach. We denote the resulting trees by SODET-UC and SODET-BC, respectively. When growing SODET trees using the imbalanced datasets in Table 3—where the problem is a binary classification task involving a rare class (i.e., the last four datasets)—we followed the corresponding approach described in Sect. 3.3. Recall that this approach is referred to as the "Rare class" (RC) approach. We denote the resulting trees by SODET-RC.

Note that in SODET trees, it is necessary to estimate the hyperparameters β and γ. As explained in Sect. 3.2, β has an initial value β_1 at the first level of the tree, which needs to be estimated. This value increases linearly, according to (5), until β reaches the value 0.5 at the level L of the tree. As also noted, γ is not estimated directly, but rather an associated parameter, the quantile Q_γ.

Initially, we considered large values for L, corresponding to the level of the tree where the value of β reaches 0.5; L up to 20, for instance. The resulting SODET trees were quite large, and in general, the results were not better than those obtained with smaller values of L. Therefore, in all our experiments with balanced datasets, we fixed L to 2, 3, 4, and 5. For imbalanced datasets, we fixed $L = 5$. For each value of L, and for each pruning method described above, we used cross-validation to estimate β_1 and Q_γ simultaneously. In the balanced datasets, we considered the values 0.4, 0.3, 0.2, and 0.1 for β_1, and 0, 0.25, 0.50, 0.75, and 1 for Q_γ. In the imbalanced datasets, we used smaller values for β_1, specifically 0.25 and 0.05, to increase the degree of oversampling. For Q_γ, we considered the extreme values 0 and 1, corresponding to no data balancing and full data balancing across tree nodes, respectively.

In our experiments, all parameters requiring estimation were determined using stratified tenfold cross-validation for all datasets, except for the Body Mass Index and Credit Card Fraud datasets, where we used fivefold cross-validation due to the smaller number of samples in the minority class.

The pseudocode for the SODET algorithm is provided in Appendix 8.1.

6.3 Results

This subsection is divided into two parts. The first part presents the results obtained on the balanced datasets described in Table 3, while the second part presents the results obtained on the imbalanced datasets from the same table.

6.3.1 Balanced datasets

The performance of the trees trained on the balanced datasets was evaluated using the misclassification error rate. For each dataset, we calculated both the cross-validation (CV) error and the test error for every tree. Based on these metrics, we ranked the trees according to their CV error across all datasets, and similarly, according to their test error. Table 7 (see Appendix 8.2) presents the mean CV ranks, mean test ranks, and the global ranks for all trees. The global rank was computed as the average of the CV and test ranks, providing a balanced view of each tree's performance in both validation and test scenarios. The trees in the table are sorted by global rank in ascending order, where a lower value indicates better overall performance. The best ranks for each column are highlighted in bold.

To further illustrate the performance of the proposed SODET models in comparison with traditional decision trees, Table 4 lists, for each dataset, the SODET and CART trees that achieved the lowest test error. For both types of trees, we report the test error and the number of leaf nodes, offering a direct comparison of predictive performance and model complexity. Additionally, for each selected SODET tree, the corresponding values of the hyperparameters β and Q_γ are included, offering insights into the degree of oversampling applied during training and the distributional balance of the data across the tree nodes.

The results presented in Tables 7 and 4 demonstrate the competitiveness of SODET models over traditional CART trees. Among SODET models, Table 7 shows that SODET-UC-L5-PM4 outperformed all others in terms of combined cross-validation and test performance. According to Table 4, SODET models outperformed or matched CART in 6 out of 7 datasets in terms of test error. Additionally, SODET trees often achieved these results with fewer or comparable numbers of leaf nodes, indicating better model efficiency. The inclusion of hyperparameters β and Q_γ in SODET models allows for controlled oversampling and better data balance across tree nodes—features not available in CART.

6.3.2 Imbalanced datasets

In the previous section, we used the misclassification error to evaluate the performance of SODET and CART trees and C5.0 trained on balanced datasets. However, in this section, we do not use misclassification error to assess the models trained on imbalanced datasets. As is well known, misclassification error is not appropriate in imbalanced scenarios, as it tends to overemphasize the performance on the majority class [2]. In imbalanced learning, the primary objective is to improve the classification of minority class instances while maintaining acceptable performance on the majority

Table 4 SODET and CART trees with the lowest test error for each dataset

Dataset	Tree	Test error (%)	# Leaf nodes	β_1	Q_γ
Banknote	SODET-UC-L2-PM4	1.53	18	0.1	1
	CART-PM4	2.84	18		
Human resources	SODET-BC-L2-PM5	1.88	22	0.4	0
	CART-PM4	1.98	57		
Iris	SODET-UC-L3-PM5	2	4	0.4	0
	CART-PM5	6	3		
Game	SODET-BC-L2-PM1	5.64	70	0.2	0.25
	CART-PM1/2/3	4.08	68		
Gene Cancer	SODET-UC-L2-PM1	2.25	10	0.3	0
	CART-PM1/2/3	2.25	10		
Rice	SODET-UC-L5-PM4	7.80	4	0.3	0.75
	CART-PM4	7.87	2		
Vehicle silhouettes	SODET-BC-L3-PM4	28.72	69	0.3	0
	CART-PM1/2/3	34.04	90		

The test error and the number of leaf nodes are shown for both types of trees. The values of the hyperparameters β and Q_γ of the SODET trees are also provided

Table 5 SODET and CART trees with the highest AUC-ROC for each dataset

Dataset	Tree	AUC-ROC	# Leaf nodes	β_1	Q_γ
Bank Loan	SODET-RC-L5-PM5	0.9721	6	0.05	0
	CART-PM5	0.9221	4		
Body Mass Index	SODET-RC-L5-PM3	0.9938	9	0.25	1
	CART-PM1/2/3	0.9513	9		
Stroke	SODET-RC-L5-PM5	0.8333	24	0.25	0
	CART-PM1/2/3	0.5829	214		
Credit Card Fraud	SODET-RC-L5-PM5	0.9688	2	0.25	1
	CART-PM5	0.9688	2		

The AUC-ROC in the test set and the number of leaf nodes are shown for both types of trees. The values of the hyperparameters β and Q_γ of the SODET trees are also provided

class. To reflect this goal, several alternative performance measures have been proposed that better capture this user preference, such as the geometric mean, the F_1 score, precision, specificity, and recall [2]. However, all of these measures require selecting an appropriate classification threshold—often not 0.5. Identifying the optimal threshold per domain would require extensive experimentation, which is beyond the scope of this study. Therefore, we adopted the Area Under the ROC Curve (AUC-ROC) as a general, threshold-independent measure of performance suitable for imbalanced learning tasks [2]. Table 5 presents the best-performing SODET and CART trees for each imbalanced dataset, based on AUC-ROC measured on the corresponding test sets. For each dataset, we report the AUC-ROC score, the number of leaf nodes, and the corresponding hyperparameter values β and Q_γ for the SODET trees.

The results shown in Table 5 highlight the superiority of SODET trees compared to CART trees when dealing with imbalanced datasets. In three out of the four datasets (Bank Loan, Body Mass Index, and Stroke), SODET trees achieved significantly higher AUC-ROC scores than their CART counterparts, indicating a stronger ability to discriminate between minority and majority class instances. Notably, in the Stroke dataset, the difference was substantial: the SODET tree achieved an AUC-ROC of 0.8333, compared to only 0.5829 for CART, while also being considerably more compact (24 vs. 214 leaf nodes). For the Credit Card Fraud dataset, both models attained the same AUC-ROC value; however, the SODET tree achieved this performance with the same number of leaf nodes, demonstrating equivalent complexity but with potential advantages stemming from its oversampling and balancing mechanisms.

These findings suggest that SODET trees not only deliver superior discriminative performance in imbalanced settings but can also yield simpler and more interpretable models, making them a robust and effective alternative to traditional decision trees.

6.3.3 Additional experiments

To conclude the empirical study, we will now use the results of these preliminary experiences in order to guide us in the choices concerning some of the "parameters" of SODET trees. We have to decide: (1) a value for β_1; (2) a value for Q_γ; (3) a value for L (level of the tree at which oversampling ends); (4) a pruning method; (5) between BC (balance the proportions per class inside the mode) and UC (leave proportions as they are). We will thus take the average ranks for all of the models used and for each value of these five choices, which can be seen in Table 7, to make some choices.

We have finally decide to use, in the next experiments, always the level $L = 5$, that is, we stop oversampling when the node reaches the level 5 of the tree. As for the pruning method, we will choose PM4 (the same as CART but without 1-SE rule). The choice between balancing the proportions per class inside each node or not seems to have little effect and so we will choose to leave the proportions as they are (UC). These choices will certainly speed up the construction of SODET trees, leaving thus only two parameters, β_1 and Q_γ, to choose using cross-validation.

Once SODET has these hyperparameters that we have to choose, 40 models in total, and since we are not using High-Performance Computing (HPC) in our experiments (although, as said above, HPC is very well appropriated for SODET), we decided to fix these hyperparameters in the next experiments and use only one of these 40 models. That is, we have fixed the hyperparameter $L = 5$ and we have only used one pruning method (the same as CART) and also, we have used the UC (unchanged class) approach in each node. For that reason, the results we present below were obtained in this restricted situation and eventually, without these restrictions, better results for SODET could be obtained.

Next, we compare experimentally our new methodology, SODET, with CART and C5.0. For CART and C5.0 we will run experiments with and without synthetic data generation. Naturally, in these two well known methodologies, synthetic data creation can only be done in the root, that is, in the entire dataset and not inside each node, as in SODET. For that reason, the "variability" of synthetic data for these two procedures is naturally smaller than the "variability" of synthetic data for SODET, since

in SODET the synthetic data creation is done, separately, in each node, that is, inside the corresponding input space region. Due to this greater variability, SODET should in principle be run a larger number of times. We decided to run three times each of these methods. CART was run without oversampling, with oversampling based on the method proposed in this paper (CART OVERSAMP), and with oversampling based on SMOTE NC (CART SMOTE NC); similarly for C5.0. The detailed results can be found in Tables 8 and 9 in the Appendices section. In Table 6, we can see the comparison of the aggregated SODET (average of the three runs), CART (average of CART, CART OVERSAMP, and CART SMOTE NC), and C5.0 (average of C5.0, C5.0 OVERSAMP, and C5.0 SMOTE NC) models on balanced datasets. Reported metrics include test error, cross-validation error, and the number of leaf nodes. The last row presents the mean performance across datasets, and the best result for each dataset and metric is highlighted in bold.

SODET achieves the lowest overall mean test error (6.64). This is particularly relevant because: test error is the most important metric (true generalization performance); it clearly outperforms aggregated CART (7.76); it shows an even larger improvement over aggregated C50 (8.15). In terms of final predictive performance, SODET is the strongest model. Now, regarding cross-validation performance, although CART presents the best average cross-validation error, SODET remains very close to C5.0, translates validation performance into better real test performance, and shows greater consistency between validation and test results compared to C5.0. In other words, CART appears slightly more optimistic during validation, but SODET converts its validation performance more effectively into actual test performance. With respect to model complexity, SODET has a complexity very close to C5.0, it is only slightly more complex than CART, and the complexity difference is small relative to the gain in test error. Hence, there is a favorable trade-off between complexity and predictive performance.

As a final comment, remark that Tables 8 and 9 in the Appendices section show the average and standard deviation of the runtime of each method during cross-validation. Naturally, the runtime of SODET is higher, which again makes it very well appropriated to an HPC platform. To further analyze computational scaling, we evaluated runtime as a function of dataset characteristics. While runtime plotted against the number of instances and features separately (see Appendix 8.3) did not exhibit consistent monotonic trends, the combined measure of dataset size, defined as the product of the number of instances and features, provides a clearer proxy for overall problem complexity. The number of instances is computed as the number of samples in each cross-validation fold plus the average number of generated synthetic samples per fold. As shown in Fig. 1, the runtime of all methods increases with dataset size, exhibiting a consistent upward trend. Among the three methods, C5.0 is the fastest, followed by CART, while SODET incurs higher computational cost. These results provide empirical evidence of how computational cost scales with problem size across the compared methods.

Table 6 Comparison of the aggregated SODET (average of the three runs), CART (average of CART, CART OVERSAMP, and CART SMOTE NC), and C5.0 (average of C5.0, C5.0 OVERSAMP, and C5.0 SMOTE NC) models on balanced datasets

Dataset	Test error (%)			CV error (%)			# leaf nodes		
	SODET	CART	C5.0	SODET	CART	C5.0	SODET	CART	C5.0
Iris	**2.67**	10	7.33	**3.67**	3.73	5.37	**5**	7.67	10.33
Vehicle silhouettes	**32.27**	34.51	32.74	25.84	**19.18**	21.17	85.33	**65**	78.33
Banknote	**1.82**	2.84	1.90	0.91	0.86	**0.68**	28.67	**19.33**	23
Rice	7.87	8.32	**7.69**	7.18	**6.62**	6.87	**4.67**	14	5
Sleep	**5.65**	**5.65**	6.18	**5.60**	5.74	6.82	5.67	**4**	7.33
Game	5.12	**4.29**	10.03	6.42	**6.18**	11.11	60.67	60	**51.33**
Kidney disease	**0**	**0**	**0**	3.82	3.19	**1.60**	**2**	**2**	5
Human resources	2.08	**1.94**	2.92	**2.31**	2.48	2.80	19.67	**17**	20.33
Gene Cancer	**2.25**	**2.25**	4.37	1.13	**0.98**	1.19	**10**	**10**	13
Mean	**6.64**	7.76	8.13	6.32	**5.44**	6.40	24.63	**22.11**	23.74

Reported metrics include test error, cross-validation error, and the number of leaf nodes. The last row presents the mean performance across datasets, and the best result for each dataset and metric is highlighted in bold

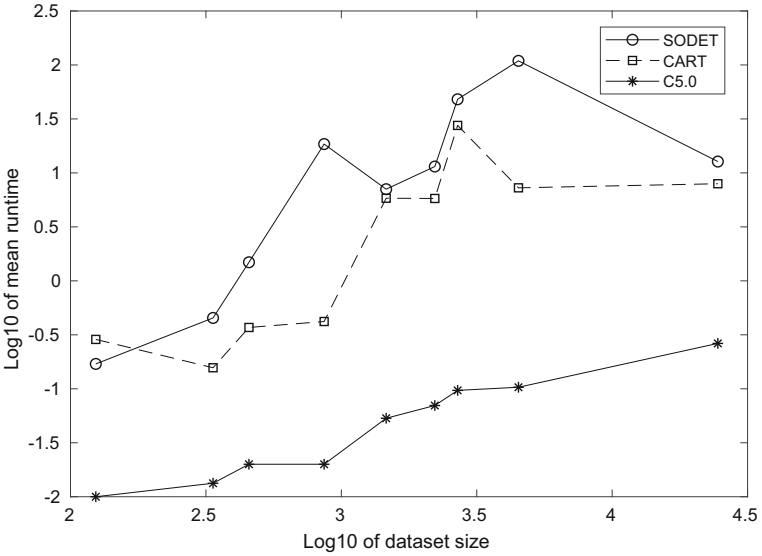

Fig. 1 Mean runtime as a function of dataset size ($\log_{10}$ scale in both axes). Results correspond to the aggregated SODET (average of the three runs), CART (average of CART, CART OVERSAMP, and CART SMOTE NC), and C5.0 (average of C5.0, C5.0 OVERSAMP, and C5.0 SMOTE NC) models on balanced datasets

7 Conclusions and future work

The main contribution of this work is a novel method to build decision trees, SODET—synthetic oversampling decision trees. The main novelty consists in finding more informative splits of the nodes by reducing the statistical noise in the estimation of the split. As we go down an usual decision tree, the number of samples in each node reduces significantly, which makes machine learning models sensitive. In fact, the fewer samples we have, the higher the variance of that split estimate. By creating more effective synthetic data points, we can stabilize the split estimate and reduce its variance. These synthetic points between real samples, force the model to learn a boundary that generalizes better instead of overfitting to the exact position of the original points. It acts thus as a sort of regularization, encouraging better generalization.

Our method is appropriate in any circumstances, both with balanced and unbalanced data and for any type of predictive variables. SODET fights the greedy nature of decision trees by creating, inside each internal node of the tree, synthetic observations for the corresponding input region only and not the entire predictive input space, as other methods that use oversampling do. As a result, we can profit from the synthetic data creation not only to have more observations in each node but also to have more balanced splits and more balanced classes. The experimental results seem to indicate that SODET trees are competitive when compared to CART and C5.0 models. On balanced datasets, SODET models achieved in general lower test errors, sometimes with simpler tree structures. In imbalanced scenarios, SODET trees delivered in general higher AUC-ROC scores, showing improved ability to correctly classify minority class instances while maintaining or improving model compactness.

Overall, SODET offers an effective and robust approach to decision tree learning, combining enhanced predictive performance with improved control over model complexity and class balance.

Although our experiments were conducted on a standard computing platform, the proposed decision tree methodology is inherently parallelizable. Each node in the tree can perform oversampling, distance computation, and splitting independently, which aligns naturally with distributed-memory and shared-memory parallel architectures. As data sizes and dimensionality continue to increase in many application domains, implementing this approach within an HPC environment could substantially reduce training time and improve scalability. Since SODET generates synthetic samples, its results are inherently stochastic, so to further strengthen our experiments we plan in the future to carry out an additional robustness study with more runs, to support a more reliable generalization.

In order to reinforce the conclusions present in this manuscript, we plan to do in the future a larger experimental study, with more datasets, so that we can perform a statistical comparison between the final results using for instance Friedman's with the Nemenyi post hoc test. For that we will need a larger number of datasets, of the same type, so that the results of the nonparametric test make sense.

We are now planning to extend this work to the context of regression trees. In addition, we plan to use the idea of SODET and merge it with Random Forests, which combines multiple decision trees trained on random subsets of data (bagging) and random subsets of features. Instead of using random subsets of data, for each random subset of features, we will build a SODET tree, which already uses inside its nodes random data, generated synthetically.

8 Appendices

8.1 Pseudocode for the SODET algorithm

Algorithm 1: SODET

Input: Training data (X, Y); CV folds $\mathcal{F}$; candidate sets $\mathcal{B}$ for β_1 and $\mathcal{Q}$ for Q_γ; level L; balance flag; pruning method
Output: Optimal hyperparameters (β_1, Q_γ) and corresponding pruned tree T_p
Set random seed;
foreach $\beta_1 \in \mathcal{B}$ **do**
  foreach $Q_\gamma \in \mathcal{Q}$ **do**
    foreach *fold* $f \in \mathcal{F}$ **do**
      Split $(X, Y) \rightarrow (X_{tr}, Y_{tr}), (X_{val}, Y_{val})$;
      $T \leftarrow$ BUILDSODET$(X_{tr}, Y_{tr}, \beta_1, Q_\gamma, L, 1,$ balance flag$)$;
      $T_p \leftarrow$ prune T;
      $\hat{Y}_{val} \leftarrow T_p(X_{val})$;
      Compute validation error between Y_{val} and $\hat{Y}_{val}$;
    Compute mean CV error over folds;
Select optimal (β_1, Q_γ) minimizing CV error;
$T \leftarrow$ BUILDSODET$(X, Y, \beta_1, Q_\gamma, L, 1,$ balance flag$)$;
$T_p \leftarrow$ prune T;
return optimal (β_1, Q_γ) and T_p;

Algorithm 2: BUILDSODET (Recursively)

Input: Node data (X, Y); β_1; Q_γ; level L; current level l; balance flag
Output: Tree node
if *all labels in Y are identical* **then**
  return leaf node with class label;
else
  Select best split (j, s) using Gini index (feature j, threshold s);
  if *no valid split* **then**
    return leaf node with majority class label;
  Split $(X, Y) \rightarrow (X_L, Y_L), (X_R, Y_R)$;
  Compute proportions p_L, p_R;
  Compute $\beta(l)$ from β_1 and L (with $\beta(l) = 0.5$ for $l \geq L$);
  Compute $\gamma_{min}, \gamma_{max}$;
  Compute $\gamma \in [\gamma_{min}, \gamma_{max}]$ from Q_γ;
  Compute total synthetic samples $n_{syn} = (1 - 2\beta(l))\,|Y|$;
  Compute left and right synthetic samples n_{syn_L} and n_{syn_R};
  `// Allocate synthetic samples`
  if *balance flag* **then**
    Allocate n_{syn_L} and n_{syn_R} across classes to balance class distribution;
  else
    Allocate n_{syn_L} and n_{syn_R} proportionally to class frequencies;
  `// Generate synthetic samples (randomized)`
  foreach *child node* $d \in \{L, R\}$ **do**
    foreach *class* c **do**
      Compute class-wise synthetic samples $n^c_{syn_d}$;
      for $i = 1$ **to** $n^c_{syn_d}$ **do**
        Randomly select seed sample x in X_d belonging to class c;
        Determine nearest neighbors of x in X_d belonging to class c;
        Generate synthetic sample by combining x with its neighbors;
  Augment (X_L, Y_L) and (X_R, Y_R) with synthetic data;
  $left \leftarrow$ BUILDSODET$(X_L, Y_L, \beta_1, Q_\gamma, L, l + 1,$ balance flag$)$;
  $right \leftarrow$ BUILDSODET$(X_R, Y_R, \beta_1, Q_\gamma, L, l + 1,$ balance flag$)$;
  return node$(j, s, left, right)$;

8.2 Tables

See Tables 7, 8, 9.

Table 7 Mean ranks of the trees in balanced datasets, calculated based on the cross-validation error, CV rank, and the test error, Test rank

Tree	CV rank	Test rank	Global rank	Tree	CV rank	Test rank	Global rank
SODET-UC-L5-PM4	**6.29**	15.57	**10.93**	SODET-UC-L2-PM5	24.43	21.07	22.75
SODET-UC-L3-PM4	10.00	14.79	12.39	SODET-BC-L2-PM3	22.14	24.07	23.11
SODET-UC-L2-PM4	11.29	18.21	14.75	SODET-UC-L2-PM2	24.29	22.14	23.21
SODET-UC-L5-PM3	15.36	**14.43**	14.89	SODET-BC-L3-PM3	24.07	23.36	23.71
SODET-BC-L5-PM4	13.57	16.50	15.04	SODET-BC-L4-PM2	25.07	22.86	23.96
SODET-BC-L4-PM4	15.50	14.86	15.18	SODET-BC-L2-PM1	21.93	26.71	24.32
SODET-UC-L5-PM1	15.50	17.50	16.50	SODET-UC-L2-PM1	21.57	28.29	24.93
SODET-BC-L5-PM1	16.50	16.79	16.64	SODET-UC-L5-PM5	26.43	23.86	25.14
SODET-UC-L4-PM4	11.50	22.71	17.11	SODET-BC-L5-PM2	24.21	26.57	25.39
SODET-BC-L2-PM4	13.29	23.50	18.39	SODET-BC-L3-PM1	25.07	25.93	25.50
SODET-UC-L5-PM2	19.43	19.14	19.29	SODET-UC-L4-PM1	22.21	28.93	25.57
SODET-BC-L3-PM4	15.64	23.43	19.54	SODET-UC-L4-PM2	24.43	27.50	25.96
SODET-UC-L2-PM3	21.14	18.43	19.79	SODET-UC-L3-PM5	30.21	22.93	26.57
SODET-UC-L3-PM3	18.14	21.43	19.79	SODET-BC-L3-PM2	26.07	27.21	26.64
SODET-UC-L4-PM3	22.07	17.79	19.93	SODET-BC-L5-PM5	30.79	23.07	26.93
SODET-BC-L4-PM3	22.21	19.00	20.61	SODET-BC-L2-PM5	30.07	24.93	27.50
SODET-BC-L4-PM1	21.00	20.43	20.71	CART-PM5	30.57	25.71	28.14
CART-PM4	23.57	18.29	20.93	SODET-BC-L3-PM5	34.29	22.36	28.32
SODET-UC-L3-PM1	17.86	24.57	21.21	CART-PM1/2/3	38.57	19.36	28.96
SODET-BC-L2-PM2	23.71	18.86	21.29	SODET-BC-L4-PM5	33.79	25.07	29.43
SODET-BC-L5-PM3	16.00	26.64	21.32	SODET-UC-L4-PM5	35.29	28.57	31.93
SODET-UC-L3-PM2	20.93	22.64	21.79				

The Global rank is the average of the CV rank and the Test rank. The trees are ordered based on the Global rank. The minimum ranks are in bold

Table 8 Detailed performance results of SODET-UC-L5, CART (including CART, CART OVERSAMP and CART SMOTE NC), and C5.0 (including C5.0, C5.0 OVERSAMP and C5.0 SMOTE NC) across the first five balanced datasets

Dataset	Model	Test error (%)	CV error (%)	# leaf nodes	β_1	Q_γ	Runtime (sec)
	SODET-UC-L5	2	3 ± 6.75	5	0.1	0	0.25 ± 0.03
	SODET-UC-L5	2	4 ± 8.43	4	0.3	0.5	0.12 ± 0.01
	SODET-UC-L5	4	4 ± 6.99	6	0.3	0	0.14 ± 0.01
Iris	CART	10	7 ± 9.49	5			0.05 ± 0.01
	CART OVERSAMP	10	2.9 ± 3.55	9			0.42 ± 0.04
	CART SMOTE NC	10	1.29 ± 2.26	9			0.39 ± 0.04
	C5.0	6	9 ± 9.94	8			0.01 ± 0.00
	C5.0 OVERSAMP	6	3.87 ± 3.97	12			0.01 ± 0.00
	C5.0 SMOTE NC	10	3.23 ± 4.02	11			0.01 ± 0.01
Dataset	Model	Test error (%)	CV error (%)	# leaf nodes	β_1	Q_γ	Runtime (sec)
	SODET-UC-L5	31.91	25.18 ± 4.81	72	0.3	1	7.03 ± 0.25
	SODET-UC-L5	32.98	26.08 ± 4.75	52	0.4	0.75	4.67 ± 0.13
	SODET-UC-L5	31.91	26.25 ± 4.28	132	0.1	0.25	22.67 ± 0.62
Vehicle	CART	35.46	25.88 ± 2.26	15			2.82 ± 0.40
Silhouettes	CART OVERSAMP	34.04	18.28 ± 3.03	90			7.85 ± 0.26
	CART SMOTE NC	34.04	13.38 ± 2.61	90			6.72 ± 0.22
	C5.0	28.01	30.14 ± 4	50			0.03 ± 0.01
	C5.0 OVERSAMP	35.46	18.84 ± 2.51	105			0.09 ± 0.02
	C5.0 SMOTE NC	34.75	14.52 ± 1.49	80			0.09 ± 0.01
Dataset	Model	Test error (%)	CV error (%)	# leaf nodes	β_1	Q_γ	Runtime (sec)
	SODET-UC-L5	2.19	0.77 ± 0.9	29	0.1	0	8.89 ± 0.34
	SODET-UC-L5	1.53	0.98 ± 0.81	30	0.1	0.5	8.21 ± 0.25
	SODET-UC-L5	1.75	0.98 ± 0.96	27	0.3	1	4.09 ± 0.13

Table 8 continued

Dataset	Model	Test error (%)	CV error (%)	# leaf nodes	β_1	Q_γ	Runtime (sec)
Banknote	CART	2.84	1.64 ± 0.93	18			2.02 ± 0.12
	CART OVERSAMP	2.84	0.57 ± 0.54	20			8.17 ± 0.40
	CART SMOTE NC	2.84	0.38 ± 0.32	20			7.29 ± 0.55
	C5.0	2.19	1.32 ± 1.35	19			0.02 ± 0.00
	C5.0 OVERSAMP	1.31	0.44 ± 0.32	26			0.07 ± 0.02
	C5.0 SMOTE NC	2.19	0.27 ± 0.32	24			0.07 ± 0.01
Dataset	Model	Test error (%)	CV error (%)	# leaf nodes	β_1	Q_γ	Runtime (sec)
	SODET-UC-L5	7.95	7.09 ± 1.08	8	0.3	0.25	48.25 ± 1.79
	SODET-UC-L5	7.8	7.2 ± 0.87	4	0.3	0.75	58.43 ± 1.21
	SODET-UC-L5	7.87	7.24 ± 0.91	2	0.4	0.25	37.6 ± 1.05
Rice	CART	7.87	7.28 ± 0.93	2			22.54 ± 0.43
	CART OVERSAMP	9.29	6.38 ± 0.97	28			32.18 ± 1.12
	CART SMOTE NC	7.8	6.2 ± 1.12	12			27.88 ± 1.07
	C5.0	7.8	7.56 ± 0.83	5			0.05 ± 0.00
	C5.0 OVERSAMP	7.48	6.51 ± 0.95	5			0.11 ± 0.05
	C5.0 SMOTE NC	7.8	6.54 ± 1.08	5			0.13 ± 0.03
Dataset	Model	Test error (%)	CV error (%)	# leaf nodes	β_1	Q_γ	Runtime (sec)
	SODET-UC-L5	5.65	5.6 ± 3.86	4	0.4	0	1.2 ± 0.03
	SODET-UC-L5	5.65	5.6 ± 3.86	5	0.4	0	1.48 ± 0.08
	SODET-UC-L5	5.65	5.6 ± 3.86	8	0.4	0	1.78 ± 0.13

Table 8 continued

Dataset	Model	Test error (%)	CV error (%)	# leaf nodes	β_1	Q_γ	Runtime (sec)
Sleep	CART	5.65	6 ± 3.89	4			0.21 ± 0.01
	CART OVERSAMP	5.65	6.13 ± 2.55	4			0.56 ± 0.05
	CART SMOTE NC	5.65	5.09 ± 4.13	4			0.34 ± 0.03
	C5.0	6.45	6.8 ± 3.79	6			0.02 ± 0.01
	C5.0 OVERSAMP	5.65	7.48 ± 2.17	7			0.02 ± 0.00
	C5.0 SMOTE NC	6.45	6.17 ± 4.26	9			0.02 ± 0.00

Reported metrics include test error (%), cross-validation error (%) (mean and standard deviation), number of leaf nodes, SODET hyperparameters (β_1 and Q_γ), and runtime in seconds (mean and standard deviation)

Table 9 Detailed performance results of SODET-UC-L5, CART (including CART, CART OVERSAMP and CART SMOTE NC), and C5.0 (including C5.0, C5.0 OVERSAMP and C5.0 SMOTE NC) across the last four balanced datasets

Dataset	Model	Test error (%)	CV error (%)	# leaf nodes	β_1	Q_γ	Runtime (sec)
	SODET-UC-L5	5.02	6.26 ± 2.35	60	0.3	0.25	17 ± 0.05
	SODET-UC-L5	5.96	6.26 ± 2.57	70	0.4	0.75	7.99 ± 0.28
	SODET-UC-L5	4.39	6.74 ± 3.32	52	0.2	1	30.5 ± 0.11
Game	CART	4.08	8.13 ± 2.41	68			0.26 ± 0.01
	CART OVERSAMP	4.39	4.68 ± 1.84	56			0.58 ± 0.07
	CART SMOTE NC	4.39	5.73 ± 2.43	56			0.42 ± 0.07
	C5.0	9.72	15.2 ± 5.03	45			0.02 ± 0.00
	C5.0 OVERSAMP	14.11	9.9 ± 2.39	60			0.02 ± 0.01
	C5.0 SMOTE NC	6.27	8.23 ± 2.75	49			0.02 ± 0.00
	SODET-UC-L5	0	3.82 ± 4.94	2	0.4	0	0.57 ± 0.06
	SODET-UC-L5	0	3.82 ± 4.94	2	0.4	0	0.39 ± 0.06
	SODET-UC-L5	0	3.82 ± 4.94	2	0.4	0	0.4 ± 0.05
Kidney	CART	0	3.82 ± 4.94	2			0.08 ± 0.01
Disease	CART OVERSAMP	0	2.87 ± 3.72	2			0.28 ± 0.03
	CART SMOTE NC	0	2.87 ± 3.72	2			0.11 ± 0.01
	C5.0	0	1.91 ± 4.03	5			0.01 ± 0.00
	C5.0 OVERSAMP	0	1.44 ± 3.04	5			0.02 ± 0.01
	C5.0 SMOTE NC	0	1.44 ± 3.04	5			0.01 ± 0.00

Table 9 continued

Dataset	Model	Test error (%)	CV error (%)	# leaf nodes	β_1	Q_γ	Runtime (sec)
	SODET-UC-L5	2.1	2.28 ± 0.91	21	0.4	0.75	107.48 ± 8.02
	SODET-UC-L5	2.1	2.31 ± 0.80	22	0.4	0	108.19 ± 1.56
	SODET-UC-L5	2.04	2.34 ± 0.95	16	0.4	0	111.79 ± 5.65
Human resources	CART	1.86	2.13 ± 0.89	17			2.47 ± 0.05
	CART OVERSAMP	2.04	3.06 ± 0.52	15			10.83 ± 0.79
	CART SMOTE NC	1.92	2.26 ± 0.71	19			8.52 ± 0.66
	C5.0	2.64	2.61 ± 0.79	15			0.07 ± 0.01
	C5.0 OVERSAMP	2.94	3.22 ± 0.58	21			0.12 ± 0.02
	C5.0 SMOTE NC	3.18	2.58 ± 0.91	25			0.12 ± 0.01
	SODET-UC-L5	2.25	1.13 ± 1.32	10	0.1	1	12.84 ± 0.68
	SODET-UC-L5	2.25	1.13 ± 1.32	10	0.1	0.5	14.72 ± 0.46
	SODET-UC-L5	2.25	1.13 ± 1.32	10	0.3	0.25	10.66 ± 0.25
Gene Cancer	CART	2.25	1.31 ± 1.54	10			6.5 ± 0.19
	CART OVERSAMP	2.25	0.76 ± 0.88	10			7.77 ± 0.38
	CART SMOTE NC	2.25	0.88 ± 1.04	10			9.54 ± 1.18
	C5.0	4.12	1.69 ± 1.38	13			0.2 ± 0.01
	C5.0 OVERSAMP	4.49	0.88 ± 1.03	13			0.27 ± 0.01
	C5.0 SMOTE NC	4.49	1 ± 0.99	13			0.32 ± 0.05

Reported metrics include test error (%), cross-validation error (%) (mean and standard deviation), number of leaf nodes, SODET hyperparameters (β_1 and Q_γ), and runtime in seconds (mean and standard deviation)

8.3 Figures

See Figs. 2 and 3.

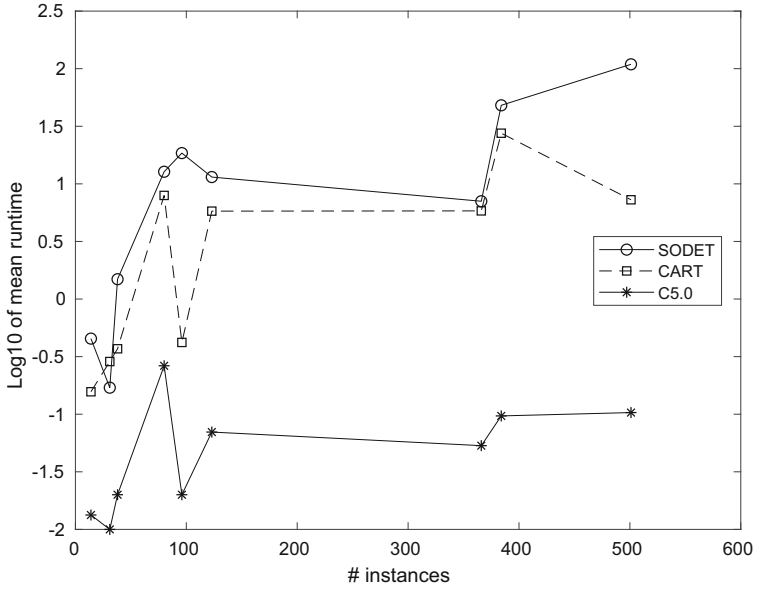

Fig. 2 Mean runtime as a function of the number of instances ($\log_{10}$ scale in the vertical axis). Results correspond to the aggregated SODET (average of the three runs), CART (average of CART, CART OVERSAMP, and CART SMOTE NC), and C5.0 (average of C5.0, C5.0 OVERSAMP, and C5.0 SMOTE NC) models on balanced datasets

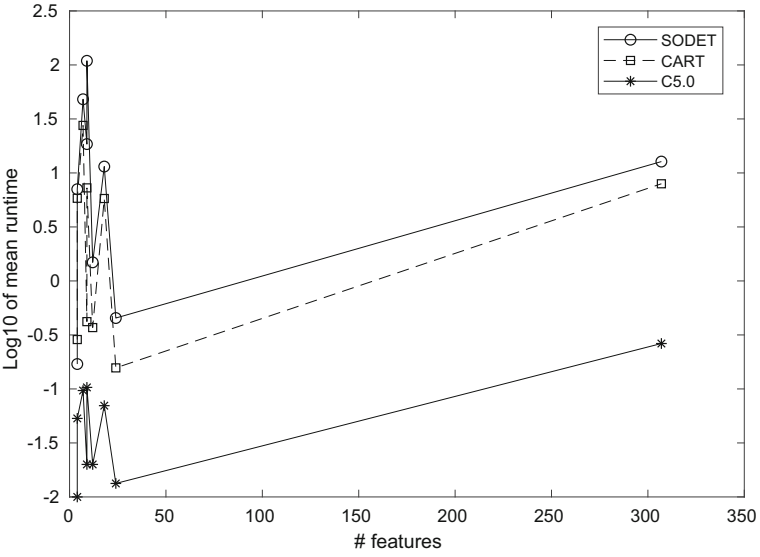

Fig. 3 Mean runtime as a function of the number of features ($\log_{10}$ scale in the vertical axis). Results correspond to the aggregated SODET (average of the three runs), CART (average of CART, CART OVERSAMP, and CART SMOTE NC), and C5.0 (average of C5.0, C5.0 OVERSAMP, and C5.0 SMOTE NC) models on balanced datasets

Acknowledgements Joaquim Fernando Pinto da Costa was partially supported by CMUP, member of LASI, which is financed by national funds through FCT, under the projects with reference UIDB/00144/2020 and UIDP/00144/2020. Hugo Alonso was partially supported by CIDMA (https://ror.org/05pm2mw36) under the Portuguese Foundation for Science and Technology (FCT, https://ror.org/00snfqn58), Grants UID/04106/2025 (https://doi.org/10.54499/UID/04106/2025) and UID/PRR/04106/2025 (https://doi.org/10.54499/UID/PRR/04106/2025). He was also funded by national funds through FCT under the Programme Contract UID/05105/2025 (https://doi.org/10.54499/UID/05105/2025).

Author Contributions J.F.P.C. contributed by conceiving the idea of the SODET trees and by developing the theoretical framework of the article. H.A. contributed by implementing the SODET trees and conducting the empirical study. Both authors contributed to the writing of the manuscript.

Funding Open access funding provided by FCT|FCCN (b-on).

Data Availability All datasets used in the empirical study are publicly available. Specifically, datasets can be accessed from the UCI Machine Learning Repository (http://archive.ics.uci.edu/) and from Kaggle (https://www.kaggle.com).

Declarations

Conflict of interest The authors declare no conflict of interest.

References

1. Alonso H, Costa J (2025) Over-sampling methods for mixed data in imbalanced problems. Commun Stat—Simul Comput 1–23
2. Branco P, Torgo L, Ribeiro R (2016) A survey of predictive modeling on imbalanced domains. ACM Comput Surv 49:1–50
3. Breiman L, Friedman J, Stone C, Olshen R (1984) Classification and regression trees. Taylor & Francis
4. Bunkhumpornpat C, Sinapiromsaran K, Lursinsap C (2012) DBSMOTE: density-based synthetic minority over-sampling technique. Appl Intell 36:664–684
5. Chawla N, Bowyer K, Hall L, Kegelmeyer W (2002) SMOTE: synthetic minority over-sampling technique. J Artif Intell Res 16:321–357
6. Chawla N (2005) Data mining for imbalanced datasets: an overview. Data Min Knowl Discov Handb 853–867
7. Cost S, Salzberg S (1993) A weighted nearest neighbor algorithm for learning with symbolic features. Mach Learn 10:57–78
8. Everitt B, Dunn G (2001) Applied multivariate data analysis, 2nd edn. Wiley
9. Gower J (1971) A general coefficient of similarity and some of its properties. Biometrics 27:857–871
10. Han H, Wang W, Mao B (2005) Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning. In: International Conference on Intelligent Computing, pp. 878–887
11. He H, Garcia E (2009) Learning from imbalanced data. IEEE Trans Knowl Data Eng 21:1263–1284
12. He H, Ma Y (2013) Imbalanced learning: foundations, algorithms, and applications. Wiley
13. He H, Bai Y, Garcia E, Li S (2008) ADASYN: adaptive synthetic sampling approach for imbalanced learning. In: Proceedings of the International Joint Conference on Neural Networks, pp 1322–1328

14. Jajuga K, Walesiak M, Bak A (2003) On the general distance measure. In: Exploratory Data Analysis in Empirical Research. Studies In Classification, Data Analysis, and Knowledge Organization, pp 104–109
15. Jolliffe I (2002) Principal component analysis, 2nd edn. Springer
16. Kaufman J, Rousseeuw P (2005) Finding groups in data: an introduction to cluster analysis. Wiley
17. Mukherjee M, Khushi M (2021) SMOTE-ENC: a novel smote-based method to generate synthetic data for nominal and continuous features. Appl Syst Innov 4:1–12
18. Podani J (1999) Extending Gower's general coefficient of similarity to ordinal characters. Taxon 48:331–340
19. Quinlan J (1993) Ross C4.5: programs for machine learning. Morgan Kaufmann
20. Ross S (2017) Introductory statistics, 4th edn. Academic Press
21. Wang F, Zheng M, Ma K, Hu X (2025) Resampling approach for imbalanced data classification based on class instance density per feature value intervals. Inf Sci 692:1–44