Chapter

Contribution to Decision Tree Induction with Python: A Review

Bouchra Lamrini

Abstract

Among the learning algorithms, one of the most popular and easiest to understand is the decision tree induction. The popularity of this method is related to three nice characteristics: interpretability, efficiency, and flexibility. Decision tree can be used for both classification and regression kind of problem. Automatic learning of a decision tree is characterised by the fact that it uses logic and mathematics to generate rules instead of selecting them based on intuition and subjectivity. In this review, we present essential steps to understand the fundamental concepts and mathematics behind decision tree from training to building. We study criteria and pruning algorithms, which have been proposed to control complexity and optimize decision tree performance. A discussion around several works and tools will be exposed to analyze the techniques of variance reduction, which do not improve or change the representation bias of decision tree. We chose *Pima Indians Diabetes* dataset to cover essential questions to understand pruning process. The paper's original contribution is to provide an up-to-date overview that is fully focused on implemented algorithms to build and optimize decision trees. This contributes to evolve future developments of decision tree induction.

Keywords: decision tree, induction learning, classification, pruning, bias-variance trade-off

1. Introduction

Decision tree induction is the most known and developed model of machine learning methods often used in data mining and business intelligence for prediction and diagnostic tasks [1–4]. It is used in classification problems, regression problems or time-dependent prediction. The main strength of decision tree induction is its interpretability characteristics. It is a graphical method designed for problems involving a sequence of decisions and successive events. More precisely, his results formalise the reasoning that an expert could have to reproduce the sequence of decisions and find a characteristic of an object. The main advantage of this type of model is that a human being can easily understand and reproduce decision sequence to predict the target category of a new instance. The results provide a graphic structure or a base of rules facilitates understanding and corresponds to human reasoning.

Learning by decision tree is part of supervised learning, where the class of each object in the database is given. The goal is to build a model from a set of examples associated with the classes to find a description for each of the classes from the common properties between the examples. Once this model has been built, we can extract a set of classification rules. In this model, the extracted rules are then used to classify new objects whose class is unknown. The classification is done by travelling a path from the root to a leaf. The class returned (default class) is the one that is most frequent among the examples on the sheet. At each internal node (decision node) of the tree, there is a test (question) which corresponds to an attribute in the learning base and a branch corresponding to each of the possible values of the attribute. At each leaf node, there is a class value. A path from the root to a node therefore corresponds to a series of attributes (questions) with their values (answers). This flowchart-like structure with recursive partitioning helps user in decision-making. It is this visualisation, which easily mimics the human-level thinking. That is why decision trees are easy to understand and interpret.

Another advantage of decision tree induction is its ability to automatically identify the most discriminating features for an use case, i.e., the most representative data inputs for a given task. This is explained by its flexibility and autonomy as a model with little assumption on the hypothesis space. It is an approach that remains particularly useful for input space problems and a powerful tool able to handle very large-scale problems, thus particularly useful in big data mining. However, it is generally less accurate than other machine learning models like neural networks.

In brief, this learning algorithm has the following three essential characteristics:

- Interpretability: Because of its flowchart-like structure, the way attributes interact to give a prediction is very readable.

- Efficiency: The induction process is done by a top-down algorithm which recursively splits terminal nodes of the current tree until they all contain elements of only one class. Practically, the algorithm is very fast in terms of running time and can be used on very large datasets (e.g. of millions of objects and thousands of features).

- Flexibility: This method does not make any hypothesis about the problem under consideration. It can handle both continuous and discrete attributes. Predictions at leaf nodes may be symbolic or numerical (in which case, trees are called regression trees). In addition, the tree induction method can be easily extended by improving tests at tree nodes (e.g. introducing linear combinations of attributes) or providing a prediction at terminal nodes by means of another model.

The review is organised into three parts. The first aims at introducing a brief history of decision tree induction. We present mathematically basics and search strategy used to train and build a decision tree. We discuss the supervised learning problem and the trade-off between a model's ability to minimise bias and variance. In this regard, we are extending our investigation to fundamental aspects, such as ensemble meta-algorithms and pruning methods, which we must put in advance for building an optimal decision tree. In the second section, we introduce some results obtained by means of the *Scikit-Learn Python* modules and *Pima Indians Diabetes* data in order to feed our discussions and our perspectives in terms of future developments and applications of Python community. The third section is devoted to the improvements of decision tree induction in order to improve its performance. We have collected some technical discussions that we raise given our experience in Research and Development (R&D). Finally, the conclusions give a general synthesis of the survey developed and discuss some ideas for future works.

2. A brief history of decision tree induction

There are many induction systems that build decision trees. Hunt et al. [5] were the first in this field to study machine learning using examples. Their concept learning system (CLS) framework builds a decision tree that tries to minimise the cost of classifying an object. There are two types of costs: (1) the cost of determining the value of a property of the object O_A exhibited by the object and (2) the misclassification cost of deciding that the object belongs to class C when its real class is K. The CLS method uses a strategy called *Lookahead* which consists of exploring the space of all possible decision trees to a fixed depth and choosing an action to minimise the cost in this limited space and then moving one level down in the tree. Depending on the depth of the *Lookahead* chosen, CLS can require a substantial amount of computation but has been able to unearth subtle patterns in the objects shown to it.

Quinlan [6] proposed Iterative Dichotomiser 3 (ID3), which takes up certain concepts of CLS. ID3 was developed following a challenge induction task on the study of end of chess games posed by Donald Michie. Analogue concept learning system (ACLS) [7] is a generalisation of ID3. CLS and ID3 require that each attribute used to describe the object takes its values in a finite set. In addition to this type of attribute, ACLS allows the use of attributes whose values can be integer. ASSISTANT [8], which is a descendant of ID3, allows the use of continuous attributes and builds a binary decision tree. ASSISTANT avoids overfitting by using a pruning technique, which has resulted in ASSISTANT-86 [9]. Another descendant of ID3 is [10, 11], which will be explained later.

There is another family of induction systems, such as the algorithm of the star AQ [12], which induces a set of decision rules from a base of examples. AQ builds an R function that covers positive examples and rejects negative ones. CN2 [13] learns a set of unordered rules of the form "IF-THEN" from a set of examples. For this, CN2 performs a top-down search (from general to specific) in the rule space, looking for the best rule, then removes the examples covered by this rule and repeats this process until no good rule is found. CN2's strategy is similar to that of AQ in that it eliminates the examples covered by the discovered rule, but it also differs in that it specialises a starting rule instead of generalising it.

Statisticians have attributed the authorship of decision tree building to Morgan and Sonquist [1], who are the first researchers to introduce the automatic interaction detector (AID) method. This method is applied to learning problems whose attribute to predict (the class) is quantitative. It works sequentially and is independent of the extent of linearity in the classifications or the order in which the explanatory factors are introduced. Morgan and Sonquist were among the first to use decision trees and among the first to use regression trees.

Several extensions have been proposed: theta AID (THAID) [2] and chi-squared AID (CHAID) [3] which uses chi-square as the independence gap to choose the best partitioning attribute. There is also a method proposed by [4] called classification and regression tree (CART) which builds a binary decision tree using the feature and threshold that yield the largest information gain at each node.

Quinlan [11] then proposes the *C4.5* algorithm for IT community. C4.5 removed the restriction that entities must be categorical by dynamically defining a discrete attribute based on numerical variables. This discretization process splits the continuous attribute value into a discrete set of intervals. C4.5 then converts the trees generated at the end of learning step into sets of if-then rules. This accuracy of each rule is well taken into account to determine the order in which they must be applied. Pruning is performed by removing the rule's precondition if the precision of the rule improves without it.

Many decision tree algorithms have been developed over the years, for example, SPRINT by Shafer et al. [14] and SLIQ by Mehta et al. [15]. One of the studies comparing decision trees and other learning algorithms was carried out by Tjen-Sien et al. [16]. The study shows that C4.5 has a very good combination of error rate and speed. C4.5 assumes that the training data is in memory. Gehrke et al. [17] proposed Rainforest, an approach to develop a fast and scalable algorithm. In [18], Kotsiantis represents a synthesis of the main basic problems of decision trees and current research work. The references cited cover the main theoretical problems that can lead the researcher into interesting directions of research and suggest possible combinations of biases to explore.

2.1 Mathematical basics and search strategy

The automatic learning of the rules in a decision tree consists in separating the learning objects into disjoint sub-samples of objects (which have no elements in common) where the majority of objects ideally have the same value for the output variable, i.e. the same class in the case of a classification problem. Each internal node performs a test on an input attribute. This test is determined automatically based on the initial training sample and according to test selection procedures that differ from one tree induction algorithm to another. For attributes with numerical values (or after encoding data), the test consists in comparing the value of an attribute with a numerical value which is called discretization threshold. According to the algorithm used, the terminal nodes of the tree are labelled either by the majority class of objects in the training sample which have reached this sheet following successive separations or by a distribution of probabilities of the classes by frequency of these objects in each class.

As indicated above, the main learning algorithms using decision trees are C4.5 [11] and CART [4]. The CART algorithm is very similar to C4.5, except for a few properties [19, 20], but it differs in that it supports numerical target variables (regression) and does not compute rule sets. The CART algorithm can be used to construct classification and regression decision trees, depending on whether the dependent variable is categorical or numeric. It also handles missing attribute values. The decision tree built by the CART algorithm is always a binary decision tree (each node has only two child nodes), also called hierarchical optimal discriminate analysis (HODA). The measurement of impurity (or purity) used in the decision tree by CART is the Gini index (C4.5 uses the notion of entropy) for classification tasks. In regression tasks, the fit method takes inputs and target arguments as in the classification setting, only that in this case target, it is expected to have floating point values (continuous values) instead of integer values. For a leaf L_i, common criteria to minimise as for determining locations for future splits are mean squared error (MSE), which minimises the $L_i + 1$ error using mean values at terminal nodes, and mean absolute error (MAE), which minimises the L_i error using median values at terminal nodes.

Several software for decision trees building are available, most of them referenced in the literature. We cite the chi-squared automatic interaction detector (CHAID) method implemented in the SIPINA[1] tool which seeks to produce a tree of limited size, allowing to initiate a data exploration. WEKA[2] uses C4.5 algorithm, and there is no need to discretize any of the attributes, and scikit-learn Python library uses an optimised version of the CART algorithm. The current (version

[1] http://eric.univ-lyon2.fr/~ricco/sipina.html

[2] https://www.cs.waikato.ac.nz/ml/weka/

0.22.1) implementation of scikit-learn library does not support categorical variables. A data encoding is mandatory at this stage (the labels transform into a value between 0 and nbClasses-1). The algorithm options are described in the Python documentation[3].

The algorithm below generally summarises the learning phase of a decision tree which begins at the top of the tree with a root node containing all the objects of the learning set:

```
Algorithm 1: _build_DT
1 if DT contains objects all of which belong to the same class then
2     return a leaf labeled with this class
3 else
4     1- Take [$a_k < a_{th}$] = _choose_test_(DT);
5     2- Split DT into $DT_{left}$ and $DT_{right}$ according to test [$a_k < a_{th}$] and build
       the sub-trees $SDT_{left}$ = build $DT_{left}$ and $SDT_{right}$ = build $DT_{right}$ from
       this sub-sets;
6     3- Creat a node with the test [$a_k < a_{th}$], make $SDT_{left}$ and $SDT_{right}$ like
       successors of this node and return the resulting tree.
7 end
8 _choose_test_(DT): Select an attribute $a_k$ and a threshold $a_{th}$ which
  minimizes the measurement of the score on DT.
```

In order for the tree to be easily interpreted, its size must be minimum. Thus, the test selection procedure applied at each node aims to choose the test (the attribute-threshold pair) which separates the objects from the current sub-sample in an optimal way, i.e. which reduces the uncertainty linked to the output variable within successor nodes. An entropy measurement (score based on a normalisation of the Shannon information measurement) allows to evaluate the gain of information provided by the test carried out. Once the model has been built, we can infer the class of a new object by propagating it in the tree from top to bottom according to the tests performed. The chosen test separates the current sample of objects into two sub-samples which are found in the successors of this node. Each test at a node makes it possible to direct any object to one of the two successors of this node according to the value of the attribute tested at this node. In other words, a decision tree is seen as a function which attributes to any object the class associated with the terminal node to which the object is directed following tests to the internal nodes of the tree. **Figure 1** illustrates an example using two input attributes with the partitioning of the input space it implies.

The induction algorithm continues to develop the tree until the terminal nodes contain sub-samples of objects that have the same output value. The label associated with a leaf in the tree is determined from the objects in the learning set that have been directed to this leaf. The majority class among the classes of these objects can be used or even the distribution of class probabilities if a stop criterion has interrupted development before reaching "pure" nodes.

The principal objective of an induction algorithm is to build on the learning data a simpler tree whose reliability is maximum, i.e. the classification error rate is minimal. However, a successful and very precise model on the learning set is not necessarily generalizable to unknown examples (objects), especially in the presence of noisy data. In this case, two sources of error expressed in the form of bias (difference between the real value and the estimated value) and the variance can generally influence the precision of a model. Several bibliographic analyses ([21]

[3] https://scikit-learn.org/stable/modules/tree.html#

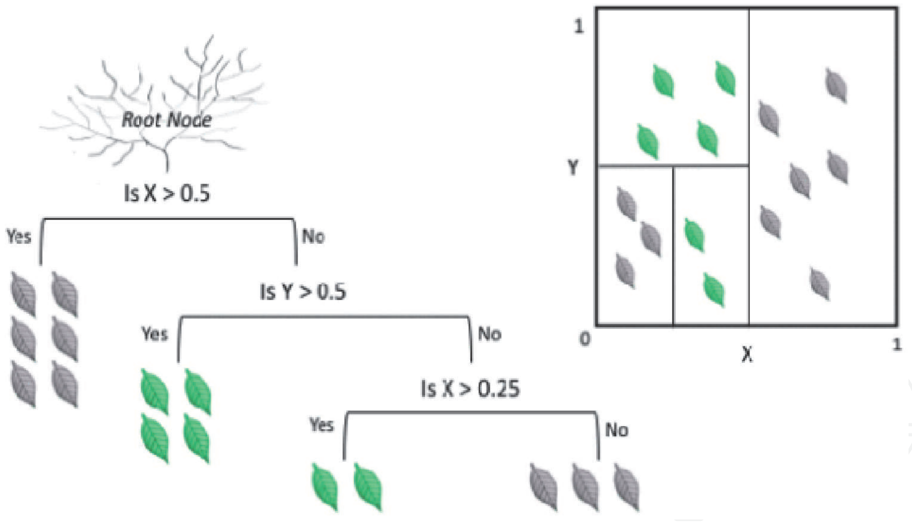


Figure 1.
An example of a decision tree and the partition it implies (Figure taken from https://www.kdnuggets.com/ website).

and the references cited in this work, [22]) have shown that decision trees suffer from a significant variance which penalises the precision of this technique. A tree may be too large due to too many test nodes determined at the bottom of the tree on sub-samples of statistically unreliable size objects. The choice of tests (attributes and thresholds) at the internal nodes of a decision tree can also depend on a sample to another which contributes to the variance of the models built. For these reasons, the criteria for stopping the development of a tree or simplification techniques such as pruning procedures is to find a good compromise between the complexity of the model and its reliability on an independent sample. These techniques can only improve the first source of error (bias) mentioned above. Different variance reduction techniques are proposed in the literature, notably the ensemble meta-algorithms such as bagging, random forests, extra-trees and boosting.

The ensemble meta-algorithms are effective in combination with decision trees. These methods differ by their way of adapting the original tree induction algorithm and/or aggregating the results. Bagging, random forests and extra-trees methods have several similarities. They independently build T constitutive trees. The predictions of different trees are aggregated as follows: each tree produces a vector of class probabilities. The T trees probability are additional in a weight vector, and the class that receives the most weight according to this one is assigned to the object. Note that these three methods use a random component and their precision can then vary slightly from one execution to another. Boosting method produces sequentially (and deterministically) the set of trees unlike these three methods using a different aggregation procedure. These methods have been successfully applied to numerous applications, notably in bioinformatics [23] and in networks [24]. Maree [22] presents a bibliographical analysis of these methods. His work covers the problem of automatic image classification using sets of random trees combined with a random extraction of sub-windows of pixel values.

2.2 Pruning

Pruning is a model selection procedure, where the models are the pruned sub-trees of the maximum tree T_0. Let $\mathcal{T}$ be the set of all binary sub-trees of T having the same root as T_0. This procedure minimises a penalised criterion where the penalty is proportional to the number of leaves in the tree [25]. Defining the optimal size of a decision tree consists in stopping the *pre-pruning* or reducing the *post-pruning* of the tree to have the best pruned sub-tree from the maximum tree to the sense of the generalisation error, i.e. improving the predictive aspect of the tree, on the one hand, and reducing its complexity, on the other hand. To this end, several pruning methods have been developed, such as:

- *Minimal cost complexity pruning* (*MCCP*), also called as *post-pruning* for the CART algorithm [4]. This method consists in constructing a nested sequence of sub-trees using a formulation called minimum cost complexity. In Section 2.2.1, we detail the general concept of this method that Scikit-Learn Library adopted in its implementation.

- *Reduced error pruning* (*REP*) consists of estimating the real error of a given sub-tree on a pruning or test set. The pruning algorithm is performed as follows: "As long as there is a tree that can be replaced by a leaf without increasing the estimate of the real error, then prune this tree". This technique gives a slightly congruent tree in the sense that some examples may be misclassified. The study of Elomaa and Kääriäinen [26] presents a detailed analysis of the REP method. In this analysis, the two authors evoke that the REP method was introduced by Quinlan [27] but the latter never presented it in an algorithmic way, which is a source of confusion. Even though REP is considered a very simple, almost trivial algorithm for pruning, many different algorithms have the same name. There is no consensus whether *REP* is a bottom-up algorithm or an iterative method. Moreover, it is not apparent that the training or pruning set is used to determine the labels of the leaves that result from pruning.

- *Pessimistic error pruning* (*PEP*). In order to overcome the disadvantages of the previous method, Quinlan [27] proposed a pruning strategy which uses a single set of construction and pruning of the tree. The tree is pruned by examining the error rate at each node and assuming that the true error rate is considerably worse. If a given node contains N records in which E among them are misclassified, then the error rate is estimated at E/N. The central concern of the algorithm is to minimise this estimate, by considering this error rate as a very optimistic version of the real error rate [28, 29].

- Minimum error pruning (MEP) was proposed by Niblett and Bratko [30], critical value pruning (CVP) by Mingers [31] and error-based pruning (EBP) proposed by Quinlan as an improvement of the PEP method, for the algorithm C4.5.

2.2.1 Pre-pruning

Pre-pruning consists in fixing a stopping rule which allows to stop the growth of a tree during learning phase by fixing a local stopping criterion which makes it possible to evaluate the informational contribution of the segmentation relating to the node that is being processed. The principle of the CHAID algorithm [32] is based on the same principle by accepting segmentation if the measure of information gain (χ^2 difference in independence or t from Tschuprow [3]) calculated on a node is significantly higher than a chosen threshold. According to Rakotomalala et al. [32, 33], formalisation involves a test of statistical hypothesis: the null hypothesis is the independence of the segmentation variable with the class attribute. If the calculated χ^2 is higher than the theoretical threshold corresponding to the risk of the first kind that we have set (respectively if the p-value calculated is lower than the risk of first kind), we accept the segmentation.

One of the cons of this algorithm is that it prematurely stops the building process of the tree. Furthermore, the use of the statistical test is considered critical. This is a classic independence test whose variable tested is produced at the end of several optimisation stages: search for the optimal discretization point for continuous variables and then search for the segmentation variable which maximises the measure used. The statistical law is no longer the same from one stage to another. The correction of the test by the introduction of certain procedures known as the *Bonferroni* correction [33] is recommended, but in practice, this type of correction does not lead to improvement in terms of classification performance. We also cite the work of [34], which proposes two pruning approaches: the first is a method of simplifying rules by the test of statistical independence to modify the pruning mechanism of the algorithm CHAID, and the second uses validation criteria inspired by the discovery technique of association rules.

The depth (maximum number of levels) of the tree and the minimum number of observations from which no further segmentation attempts are made also remain two practical options that can be fixed at start learning to manage the complexity of the model. However, the choice of these parameters remains a critical step in the tree building process because the final result depends on these parameters that we have chosen. To this is added the fact that the evaluation is local (limited to a node) and we take more account of the global evaluation of the tree. It is therefore necessary to propose a rule which is not too restrictive (respectively not too permissive) to obtain a suitable tree and not undersized (respectively not oversized).

2.2.2 Post-pruning

The algorithm for building a binary decision tree using CART browses for each node the m attributes $(x_1, x_2, \dots, x_m)$ one by one, starting with x_1 and continuing up to x_m. For each attribute, it explores all the possible tests (splits), and it chooses the best split (dichotomy) which maximises the reduction in impurity. Then, it compares the m best splits to choose the best of them. The function that measures impurity should reach its maximum when the instances are fairly distributed between the different classes and its minimum when a class contains all the examples (the node is pure). There are different functions which satisfy these properties. The function used by CART algorithm is Gini function (Gini impurity index). Gini function on a node t with a distribution of class probabilities on this node P (j|t), c = 1, ..., k is:

$$\begin{aligned} G(p) &= \phi(P(1|t), P(2|t), \dots, P(k|t)) \\ &= \sum_c P(c|t).(1 - P(c|t)) \end{aligned} \tag{1}$$

If a split s on a node t splits the subset associated with this node into two subsets, left t_G with a proportion p_G and right t_D with a proportion p_D, we can define the impurity reduction measure as follows:

$$\Delta G(s,t) = G(t) - p_G * G(t_G) - p_D * G(t_D) \tag{2}$$

On each node, if the set of candidate splits is S, the algorithm searches the best split s^* such that:

$$\Delta G(s^*, t) = max_{s \in S} \Delta G(s,t) \tag{3}$$

Suppose we got some *splits* and came up with a set of terminal nodes $\tilde{T}$. The set of *splits*, used in the same order, determines the binary tree T. We have $I(t) = G(t)p(t)$. So the impurity function on the tree is:

$$I(T) = \sum_{t \in \tilde{T}} I(t) = \sum_{t \in \tilde{T}} G(t) * p(t) \tag{4}$$

$G(t)$ is the impurity measure on the node t, and $p(t)$ is the probability that an instance belongs to the node t.

It is easy to see that the selection of the splits which maximise $\Delta_i(s,t)$ is equivalent to the selection of the splits which minimise the impurity $I(T)$ on all the trees. If we take any node $t \in \tilde{T}$ and we use a split s which partitions the node into two parts t_D and t_G, the new tree $\acute{T}$ has the following impurity:

$$I\left(\acute{T}\right) = \sum_{\tilde{T}-\{t\}} I(t) + I(t_D) + I(t_G) \tag{5}$$

Because we have partitioned the subset arrived at t and t_D and t_G, reducing the impurity of the tree is therefore:

$$I(T) - I\left(\acute{T}\right) = I(T) - I(t_D) - I(t_G) \tag{6}$$

It only depends on the node t and the *splits s*. So, to maximise the reduction of impurity in the tree on a node t, we maximise:

$$\Delta I(s,t) = I(t) - I(t_G) - I(t_D) \tag{7}$$

The proportions p_D are defined as follows: $p_D = p(t_D)/p(t)$, $p_G = p(t_G)/p(t)$ and $p_G + p_D = 1$. So, Eq. (7) can be written as follows:

$$\begin{aligned} \Delta I(s,t) &= \left[G(t) - p_G * G(t_G) - p_D * G(t_D)\right] * p(t) \\ &= \Delta G(s,t) * p(t) \end{aligned} \tag{8}$$

Since $p(t)$ is the only difference between (s,t) and (s,t), the same *splits* s^* maximises both expressions.

The stop splitting criterion used by CART was very simple: for a threshold $\beta > 0$, a node is declared terminal (leaf) if $max \Delta I(s,t) \leq \beta$. The algorithm assigns to each terminal node the most probable class.

Post-pruning is a procedure that appeared with the CART method [4]. It was very widely taken up in different forms thereafter. The principle is to build the tree in two phases. (1) The first phase of expansion consists in producing the purest possible trees and in which all segmentations are accepted even if they are not relevant. This is the principle of hurdling building. (2) In the second phase, we try to reduce the tree by using another criterion to compare trees of different sizes. The building time of the tree is of course longer. It can be penalising when the database is very large while the objective is to obtain a tree that performs better in classification phase.

The idea that was introduced by Breiman et al. [4] is to construct a sequence of trees $T_0, .., T_i, .., T_t$, which minimise a function called *cost complexity metric* (previously mentioned). This function combines two factors: the classification error rate and the number of leaves in the tree using α parameter.

For each internal node, Ne and T_0, the relationship is defined as:

$$\alpha(p) = \frac{\Delta R^S_{emp}}{|T_p| - 1} \quad (9)$$

where ΔR^S_{emp} is the number of additional errors that the decision tree makes on the set of samples S when we prune it at position p. $|p| - 1$ measures the number of sheets deleted. The tree T_{i+1} is obtained by pruning T_i at its node which has the smallest value of $\alpha(p)$ parameter. We thus obtain a sequence $T_0, .., T_i, .., T_t$ of elements of T, the last of which is reduced to a leaf. To estimate the error rate for each tree, the authors suggest using two different methods, one based on cross-validation and the other on a new test base.

3. Decision tree classifier building in Scikit-Learn

Today there are several best machine learning websites that propose tutorials to show how decision trees work using the different modules of python. We quote for example three popular websites: Towards Data Science,[4] KDnuggets,[5] and Kaggle.[6] Developers offer in a few lines of optimised code how to use decision tree method by covering the various topics concerning attribute selection measures, information gain, how to optimise decision tree performance, etc.

From our side, we choose *Pima Indians Diabetes* datasets (often used in classification problems) to examine the various tuned parameters proposed as arguments by Scikit-Learn package. The Pima are a group of Native Americans living in Arizona. A genetic predisposition allowed this group to survive normally to a diet poor of carbohydrates for years. In the recent years, a sudden shift from traditional agricultural crops to processed foods, together with a decline in physical activity, made them develop the highest prevalence of type 2 diabetes, and for this reason they have been subject of many studies. The original dataset is available at UCI Machine Learning Repository and can be downloaded from this address,[7] "diabetes-data.tar.Z", containing the distribution for 70 sets of data recorded on diabetes patients, several weeks to months worth of glucose, insulin and lifestyle data per patient. The dataset includes data from 768 women with 8 characteristics, in particular:

1. Number of times pregnant (*NTP*)

2. Plasma glucose concentration in 2 h in an oral glucose tolerance test (*PGC*)

3. Diastolic blood pressure (mm Hg) (*DBP*)

4. Triceps skinfold thickness (mm) (*TSFT*)

5. Two-hour serum insulin (mu U/ml) (*HSI*)

6. Body mass index (weight in kg/(height in m)^2) (*BMI*)

[4] https://towardsdatascience.com/decision-tree-algorithm-explained-83beb6e78ef4

[5] https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html

[6] https://www.kaggle.com/dmilla/introduction-to-decision-trees-titanic-dataset

[7] http://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes

7. Diabetes pedigree function (*DPF*)

8. Age (years)

The last column of the dataset indicates if the person has been diagnosed with diabetes or not.

Without any data preparation step (cleaning, missing values processing, etc.), we partitioned the dataset into a training data (75%) to build the tree and test data (25%) for prediction. Then we kept the default settings which we can see through the profile class function (**Figure 2**).

The Scikit-Learn documentation[8] explains in detail how to use each parameter and offers other modules and functions to search information and internal structures of classifier from training to building step. Among these parameters, we highlight in this review the following four we use to optimise the tree:

- *criterion*: Optional (default = "gini"). This parameter allows to measure the quality of a split, use the different-different attribute selection measure and supports two criteria, "gini" for the Gini index and "entropy" for the information gain.

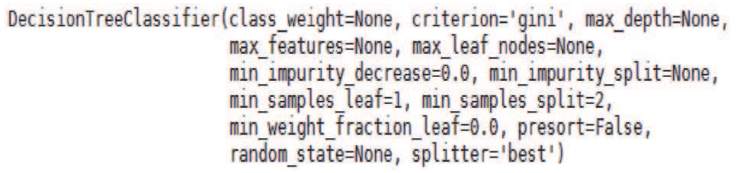
- max_depth: Optional (default = None), the maximum depth of a tree. If None, then nodes are expanded until all the leaves contain less than min_samples_split samples. A higher value of maximum depth causes overfitting, and a lower value causes underfitting.

- min_samples_leaf: Optional (default = 1), the minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.

- min_impurity_decrease: Optional (default = 0.0). A node will be split if this split induces a decrease of the impurity greater than or equal to this value. The weighted impurity decrease equation is the following:

$$N_t/N * \left(impurity - N_{tR}/N_t * right_impurity - N_{tL}/N_t * left_impurity\right) \tag{10}$$

where N is the total number of samples, N_t is the number of samples at the current node, N_{tL} is the number of samples in the left child and N_{tR} is the number of samples in the right child. N, N_t, N_{tR} and N_{tL} all refer to the weighted sum, if sample_weight is passed.



```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

Figure 2.
Default setting to create decision tree classifier without pruning.

[8] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

In this example, each internal node has a decision rule that divides the data. The node impurity is set by default at Gini ratio. A node is pure when all the objects belong to the same class, i.e. impurity = 0. The unpruned tree resulting from this setting is inexplicable and difficult to understand. In **Figures 3** and **4**, we will show you how to adjust some tuned parameters to get an optimal tree by pruning.

"export_graphviz" and "pydotplus" modules convert the decision tree classifier to a "dot" file and in "png/pdf/.." format. Using various options of this modules, you can adjust leaf colours and edit leaf content, important descriptors, etc. Personally, I really enjoyed doing it during my R&D works.

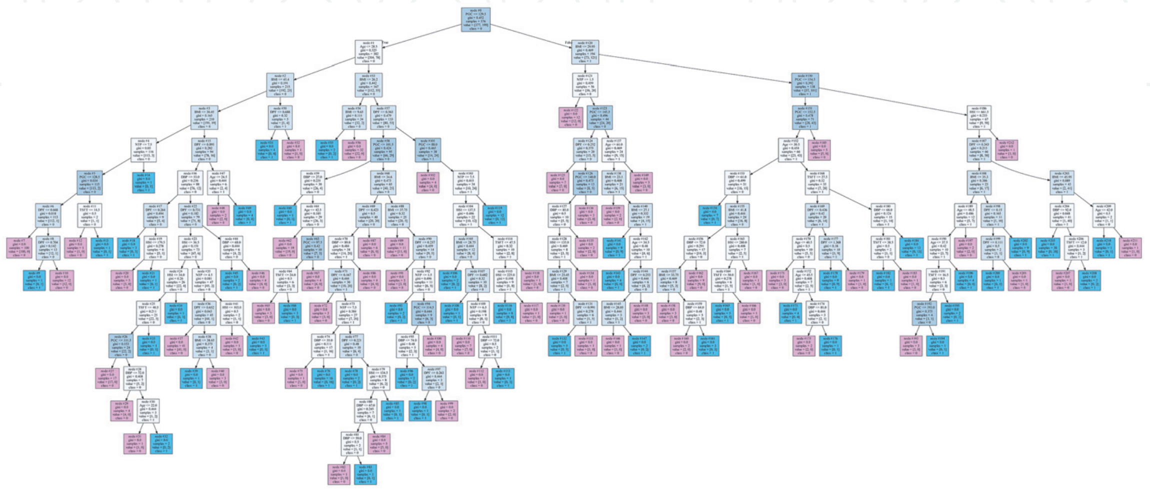


Figure 3.
Decision tree without pruning. Accuracy = 0.72.

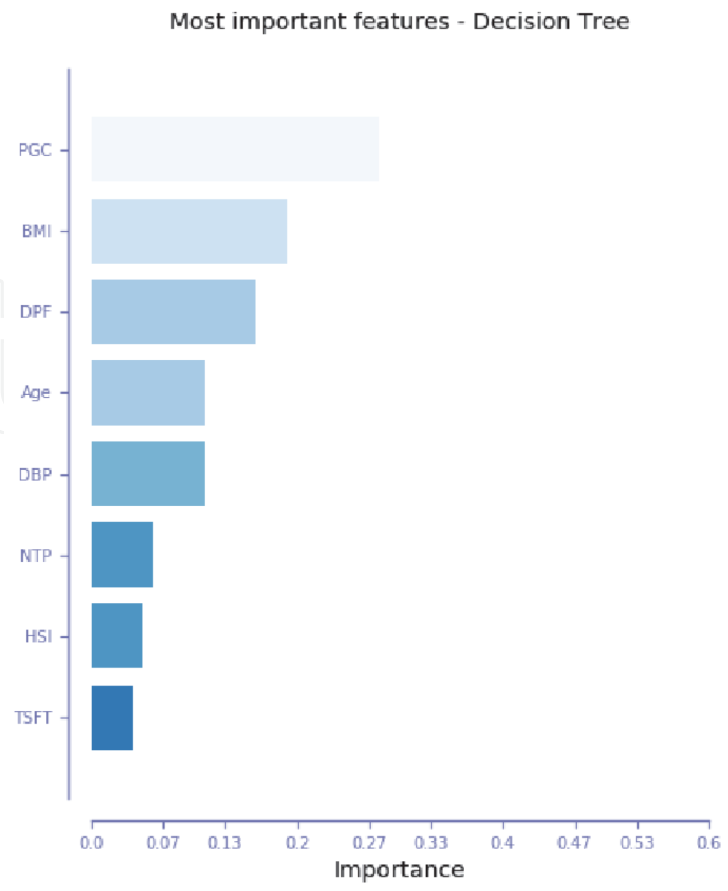


Figure 4.
Feature importance. Decision tree without pruning.

We will now adjust only one parameter, the maximum depth of the tree. This will control the tree size (number of levels). On the same data, we set maximum_depth at 4. Next, we set "min_impurity_decrease" at 0.01 and min_samples_leaf at 5. We will see that this pruned tree is less complex and easier to understand by a field expert than the previous flowchart. We will see that we have good accuracy with this setting. Accuracy can be computed by comparing actual test set values and predicted values (**Figures 5–8**).

Most important features of Pima Indians Diabetes dataset is shown in **Figures 4**, **6** and **8**. We can see the root node is glucose, which can show the glucose has the max information gain, so it confirm the common sense and the clinical

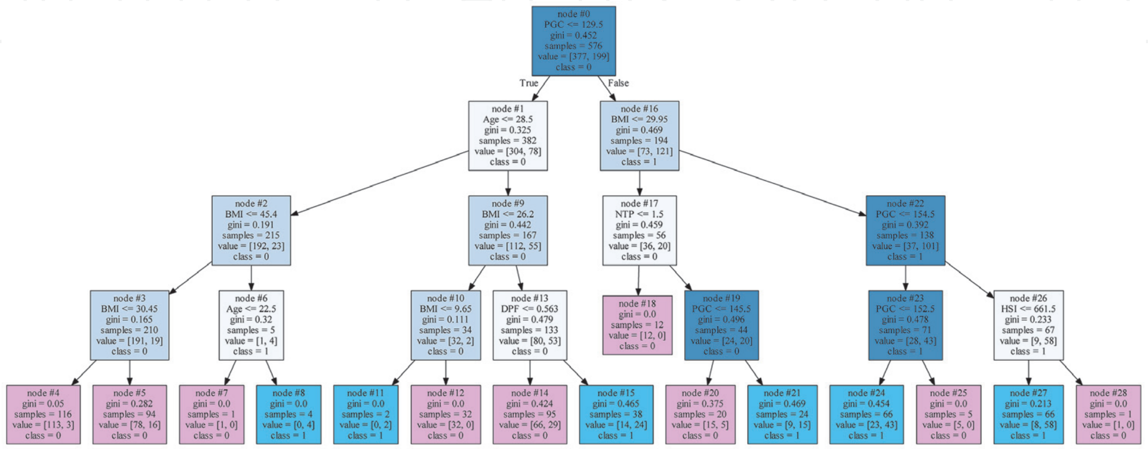


Figure 5.
Decision tree pruned by mean of maximum depth parameter. Accuracy = 0.79.

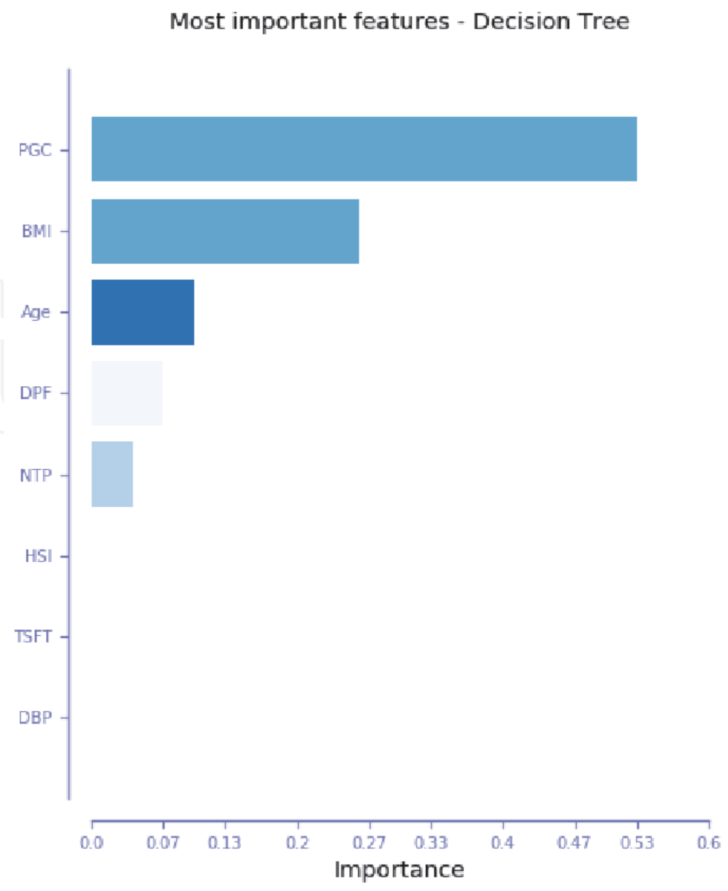


Figure 6.
*Feature importance. Decision tree after pruning (corresponding to **Figure** 5 results).*

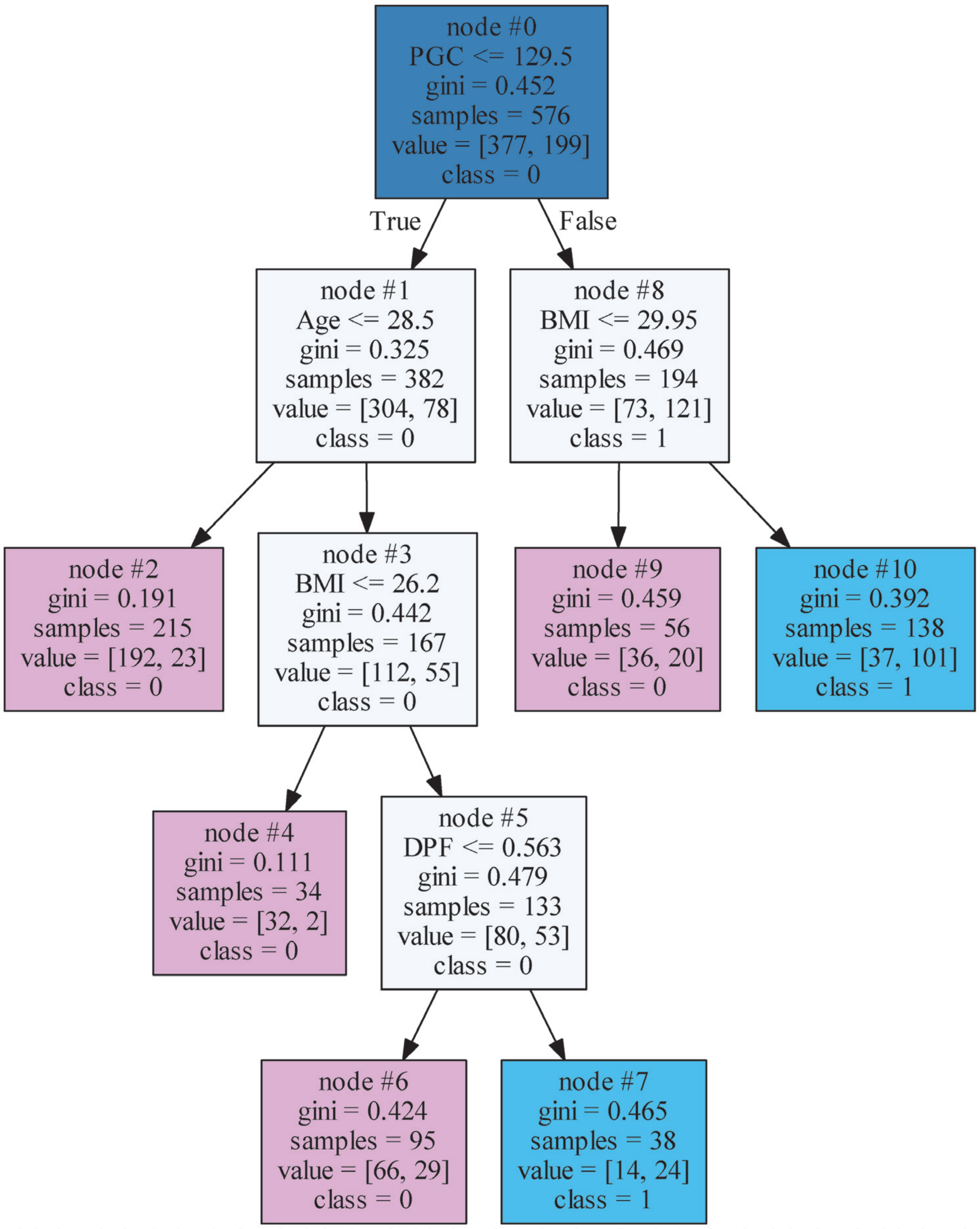


Figure 7.
Decision tree pruned by mean maximum depth and impurity parameters. Accuracy = 0.80.

diagnosis basis. Body mass index (BMI) and age are also found among the first important variables. According to consulting relevant information, we know there are three indicators to determine the diabetes mellitus, which are fasting blood glucose, random blood glucose and blood glucose tolerance. Pima Indians Diabetes dataset only has blood glucose tolerance. Prevalence of diabetes mellitus, hypertension and dyslipidaemia increase with higher BMI (BMI 25 kg/m^2). On the other hand, type 2 diabetes usually begins after age 40 and is diagnosed at an average age of 65. This is why the French National Authority for Health recommends renewing the screening test every 3 years in people over 45 years and every year if there is more than one risk factor.

Despite the stop criterion of tree depth, the trees generated may be too deep for a good practical interpretation. The notion of "accuracy" associated with each level

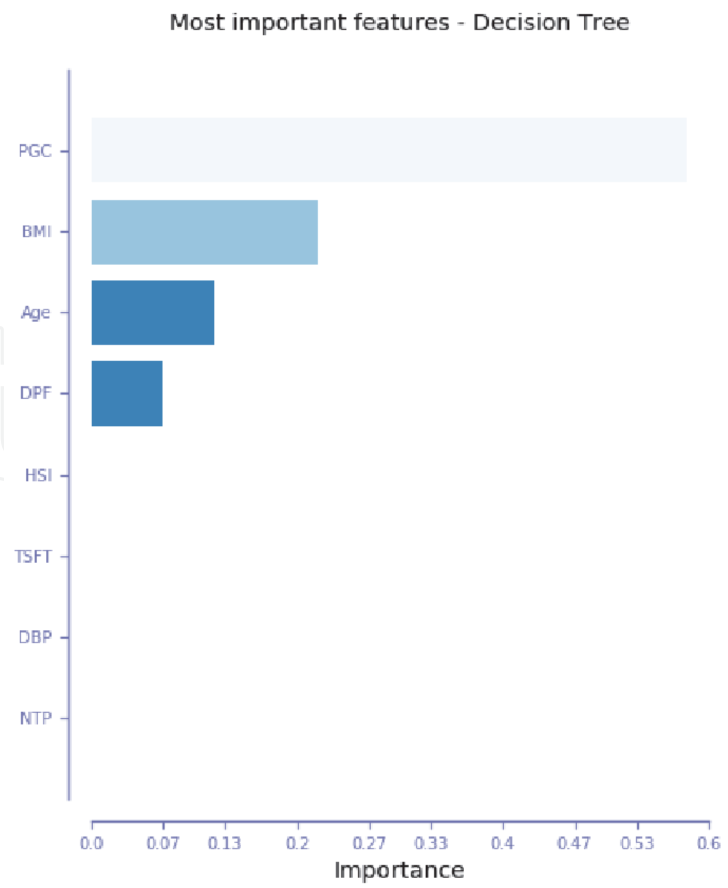


Figure 8.
*Feature importance. Decision tree after pruning (corresponding to **Figure 7** results).*

Prédit/réel	Classe A	Classe B
Classe A	VA (Vrais A)	FA (Faux A)
Classe B	FB (Faux B)	VB (Vrais B)

Table 1.
Confusion matrix.

of the tree will make it possible to present a partial tree sufficiently precise. This is based on the confusion matrix: The accuracy P is the ratio of well-classified elements to the sum of all elements and is defined by the following expression (**Table 1**):

$$\mathcal{P} = \frac{VA + VB}{VA + VB + FA + FB} \tag{11}$$

The accuracy associated with a level of the tree is calculated by summing the VA, VB, FA and FB taking into account the labels A or B of each node, and we add to VA or VB the elements corresponding to pure nodes A or B in the previous levels. We can thus decide to choose the partial tree according to the desired accuracy.

4. Discussions

Decision trees accept, like most learning methods, several hyper-parameters that control its behaviour. In our use case, we used Gini index like information criteria to split the learning data. This criterion has directed the method to build a tree with a maximum of 15 levels and to accept a node as a leaf if it includes at least five learning instances. Impurity (entropy) is a measure of disorder in dataset; if we have zero entropy, it means that all the instances of the target classes are the same, while it reaches its maximum when there is an equal number of instances of each class. At each node, we have a number of instances (from the dataset), and we measure its entropy. Setting impurity to a given value (chosen according to expertise and tests) will allow us to select the questions which give more homogeneous partitions (with the lowest entropy), when we consider only the instances for which the answer to the question is yes or no, that is to say when the entropy after answer to the question decreases.

During my previous R&D work, we used the CART algorithm implemented in the scikit-learn library. This implementation is close to the original one proposed by [4]; however there is no parameter for penalising the deviance of the model by its complexity (number of leaves) in order to build a sequence of trees nested in the prospect of optimal pruning by cross-validation. The generic function of k-fold cross-validation “GridSearchCV” can be used to optimise the depth parameter but with great precision in pruning. The depth parameter eliminates a whole level and not the only unnecessary leaves to the quality of the prediction. On the other hand, the implementation anticipates those of model aggregation methods by integrating the parameters (number of variables drawn, importance, etc.) which are specific to them. On the other hand, the graphical representation of tree is not included and requires the implementation of another free software like “Graphviz” and “Pydotplus” modules.

The pros and cons of decision trees are known and described in almost all the articles and works developed in this field. We highlight some that we consider important for industrial applications. Selecting features is an extremely important step when creating a machine learning solution. If the algorithm does not have good input functionality, it will not have enough material to learn, and the results will not be good, even if we have the best machine learning algorithm ever designed. The selection of characteristics can be done manually depending on the knowledge of the field and the machine learning method that we plan to use or by using automatic tools to evaluate and select the most promising. Another common problem with datasets is the problem of missing values. In most cases, we take a classic imputation approach using the most common value in the training data, or the median value. When we replace missing values, we should understand that we are modifying the original problem and be careful when using this data for other analytical purposes. This is a general rule in machine learning. When we change the data, we should have a clear idea of what we are changing, to avoid distorting the final results. Fortunately, decision tree requires fewer data preprocessing from users. It is used with missing data, and there is no need to normalise features. However, we must be careful in the way we describe the categorical data. Having a priori knowledge of the data field, we can favour one or more modalities of a descriptor to force the discretization process to choose a threshold, which highlights the importance of the variables. Moreover, Geurts [21] has shown that the choice of tests (attributes and thresholds) at the internal nodes of the tree can strongly depend on samples, which also contributes to the variance of the models built according to this method.

Decision tree can easily capture nonlinear patterns, which is important in big data processing. Nevertheless it is sensitive to noisy data, and it can overfit it. In big data mining, online data processing is subject to continuous development (upgrade, environment change, catching up, bugs, etc.) impacting the results expected by customers and users. To this, the problem of variation that can be reduced by bagging and boosting algorithms (that we mentioned in Section 2.1) is added.

Decision tree is biased with imbalance dataset. It is recommended to balance dataset before training to prevent the tree from being biased towards the classes that are dominant. According to scikit-learn documentation "class balancing can be done by sampling an equal number of samples from each class, or preferably by normalising the sum of the sample weights (sample_weight) for each class to the same value. Also note that weight-based pre-pruning criteria, such as min_weight_fraction_leaf, will then be less biased towards dominant classes than criteria that are not aware of the sample weights, like min_samples_leaf".

5. Conclusions

Decision trees simply respond to a classification problem. Decision tree is one of the few methods that can be presented quickly, without getting lost in mathematical formulations difficult to grasp, to hearing not specialised in data processing or machine learning. In this chapter, we have described the key elements necessary to build a decision tree from a dataset as well as the pruning methods, pre-pruning and post-pruning. We have also pointed to ensemble meta-algorithms as alternative for solving the variance problem. We have seen that letting the decision tree grow to the end causes several problems, such as overfitting. In addition, the deeper the tree is, the more the number of instances (samples) per leaf decreases. On the other hand, several studies have shown that pruning decreases the performance of the decision tree in estimating probability.

Decision tree properties are now well known. It is mainly positioned as a reference method despite the fact that efforts to develop the method are less numerous today. The references cited in this chapter are quite interesting and significant. They provide a broad overview of statistical and machine learning methods by producing a more technical description pointing the essential key points of tree building. In spite of the fact that the CART algorithm has been around for a long time, it remains an essential reference, by its precision, its exhaustiveness and the hindsight which the authors, developers and researchers demonstrate in the solutions they recommend. Academic articles also suggest new learning techniques and often use it in their comparisons to locate their work, but the preferred method in machine learning also remains C4.5 method. The availability of source code on the web justifies this success. C4.5 is now used for Coronavirus Disease 2019 (COVID-19) diagnosis [35, 36].

Finally, we would like to emphasise that the interpretability of a decision tree is a factor which can be subjective and whose importance also depends on the problem. A tree that does not have many leafs can be considered easily interpretable by a human. Some applications require good interpretability, which is not the case for all prediction applications. On industrial problems, an interpretable model with great precision is often necessary to increase knowledge of the field studied and identify new patterns that can provide solutions to needs and to several expert questions. We continue to put a lot of effort (scientific researchers, developers, experts, manufacturers, etc.) to make more improvements to this approach: decision tree induction. This chapter opens several opportunities in terms of algorithms and in terms of applications. For our use case, we would like to have more data to predict the type of diabetes and determine the proportion of each indicator, which can improve the accuracy of predicting diabetes.

Author details

Bouchra Lamrini
Ex-researcher (Senior R&D Engineer) within Livingobjects Company, Toulouse, France

*Address all correspondence to: lamrini.bouchra@gmail.com

References

[1] Morgan J, Sonquist J. Problems in the analysis of survey data, and a proposal. Journal of the American Statistical Association. 1963;**58**(2):415-435

[2] Morgan J, Messenger R. THAID-A Sequential Analysis Program for the Analysis of Nominal Scale Dependent Variables. Ann Arbor: Survey Research Center, Institute for Social Research, University of Michigan; 1973

[3] Kass G. An exploratory technique for investigating large quantities of categorical data. Applied Statistics. 1973; **29**(2):119-127

[4] Breiman L, Friedman J, Stone C, Olshen R. Classification and Regression Trees. Taylor & Francis;; 1984. Available from: https://books.google.fr/books?id=JwQx-WOmSyQC

[5] Hunt E, Marin J, Stone P. Experiments in Induction. New York, NY, USA: Academic Press; 1997. Available from: http://www.univ-tebessa.dz/fichiers/mosta/544f77fe0cf29473161c8f87.pdf

[6] Quinlan JR. Discovering rules by induction from large collections of examples. In: Michie D, editor. Expert Systems in the Micro Electronic Age. Vol. 1. Edinburgh University Press; 1979. pp. 168-201

[7] Paterson A, Niblett T. ACLS Manual. Rapport Technique. Edinburgh: Intelligent Terminals, Ltd; 1982

[8] Kononenko I, Bratko I, Roskar E. Experiments in Automatic Learning of Medical Diagnostic Rules. Technical Report. Ljubljana, Yugoslavia: Jozef Stefan Institute; 1984

[9] Cestnik B, Knononenko I, Bratko I. Assistant86-A knowledge elicitation tool for sophisticated users. In: Bratko I, Lavrac N, editors. Progress in Machine Learning. Wilmslow, UK: Sigma Press; 1987. pp. 31-45

[10] Quinlan JR, Compton PJ, Horn KA, Lazarus L. Inductive knowledge acquisition: A case study. In: Proceedings of the Second Australian Conference on Applications of Expert Systems. Boston, MA, USA: Addison Wesley Longman Publishing Co., Inc.; 1987. pp. 137-156

[11] Quinlan R. C4.5-Programs for Machine Learning. San Francisco, CA, USA: Morgan Kaufman; 1993. p. 368

[12] Michalski RS. On the quasi minimal solution of the general covering problem. In: Proceedings of the 5th International Symposium on Information Processing; 1969. pp. 125-128

[13] Clark P, Niblett T. The cn2 induction algorithm. Machine Learning. 1989;**3**(4):261283

[14] Shafer J, Agrawal R, Mehta M. SPRINT-A scalable parallel classifier for data mining. Proceeding of the VLDB Conference. 1996;**39**:261-283. DOI: 10.1007/s10462-011-9272-4

[15] Mehta M, Agrawal R, Riassnen J. SLIQ-A fast scalable classifier for data mining. Extending Database Technology. 1996;**39**:18-32

[16] Tjen-Sien L, Wei-Yin L, Yu-Shan S. SLIQ. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. Machine Learning. 2000;**40**:203-228

[17] Gehrke J, Ramakrishnan R, Ganti V. RainForest—A framework for fast decision tree construction of large datasets. Data Mining and Knowledge Discovery. 2000;**4**(2–3):127-162. DOI: 10.1023/A:1009839829793

[18] Kotsiantis SB. Decision trees—A recent overview. Artificial Intelligence Review. 2013;**39**:261-283. DOI: 10.1007/s10462-011-9272-4

[19] Rokach L, Maimon O. Top-down induction of decision trees classifiers—A survey. IEEE Transactions on Systems, Man, and Cybernetics: Part C. 2005;**35**(4):476-487. DOI: 10.1109/TSMCC.2004.843247

[20] Brijain MR, Patel R, Kushik MR, Rana K. International Journal of Engineering Development and Research. 2014;**2**(1):1-5. DOI: 10.1109/TSMCC.2004.843247

[21] Geurts P. Contributions to Decision Tree Induction: Bias/Variance Tradeoff and Time Series Classification. Belgium: University of Liège; 2002. p. 260. Available from: http://www.montefiore.ulg.ac.be/services/stochastic/pubs/2002/Geu02

[22] Marée R, Geurts P, Visimberga G, Piater J, Wehenkel L. A comparison of generic machine learning algorithms for image classification. In: Proceeding Research and Development in Intelligent Systems XX; 2004. pp. 169-182

[23] Geurts P, Fillet M, de Seny D, Meuwis MA, Merville MP, Wehenkel L. Proteomic mass spectra classification using decision tree based ensemble methods. Bioinformatics. 2004;**21**(14):3138-3145. DOI: 10.1093/bioinformatics/bti494

[24] Geurts P, Khayat E, Leduc G. A machine Learning approach to improve congestion control over wireless computer networks. In: Proceedings of the IEEE International Conference on Data Mining (ICDM'04); 2004. Available from: https://ieeexplore.ieee.org/document/1410316

[25] Genuer R, Poggi JM. Arbres CART et Forêts aléatoires, Importance et sélection de variables. hal-01387654v2; 2017. pp. 1-5. Available from: https://hal.archives-ouvertes.fr/hal-013876

[26] Elomaa T, Kääriäinen M. An analysis of reduced error pruning. Journal of Artificial Intelligence Research. 2001:163-187. Available from: http://dl.acm.org/citation.cfm?id=26079.26082

[27] Quinlan R. Simplifying decision trees. International Journal of Human Computer Studies. 1999;**51**(2):497-510

[28] Berry MJ, Linoff G. Data Mining Techniques for Marketing, Sales, and Customer Relationship Management. John Wiley & Sons; 1997. Available from: http://hdl.handle.net/2027/mdp.39015071883859

[29] Brostaux Y. Étude du Classement par Forêts Aléatoires D'échantillons Perturbés à Forte Structure D'interaction. Belgium: Faculté Universitaire des Sciences Agronomiques de Gembloux; 2005. p. 168. Available from: http://www.montefiore.ulg.ac.be/services/stochastic/pubs/2002/Geu02

[30] Niblett T, Bratko I. Learning decision rules in noisy domains. In: Proceedings of Expert Systems '86, The 6th Annual Technical Conference on Research and Development in Expert Systems III. Cambridge University Press; 1987. pp. 25-34. Available from: http://dl.acm.org/citation.cfm?id=26079.26082

[31] Mingers J. Experts systems-rule induction with statistical data. Journal of the Operational Research Society. 1987;**38**(1):39-47

[32] Rakotomalala R, Lallich S. Construction d'arbres de décision par optimisation. Revue des Sciences et Technologies de l'Information - Série RIA: Revue d'Intelligence Artificielle. 2002;**16**(6):685-703. Available from: https://hal.archives-ouvertes.fr/hal-00624091

[33] Rakotomalala R. Arbres de décision. Revue Modular. 2005;**33**:163-187. Available from: https://www.rocq.inria.fr/axis/modulad/archives/numero-33/tutorial-rakotomalala-33/rakotomalala-33-tutorial.pdf

[34] Mededjel M, Belbachir H. Post-élagage indirect des arbres de décision dans le data mining. In: 4th International Conference: Sciences of Electronic, Technologies of Information and Telecommunications; 2007. pp. 1-7. Available from: http://www.univ-tebessa.dz/fichiers/mosta/544f77fe0cf29473161c8f87.pdf

[35] Wiguna W, Riana D. Diagnosis of Coronavirus Disease 2019 (COVID-19) surveillance using C4.5 Algorithm. Jurnal Pilar Nusa Mandiri. 2020;**16**(1):71-80. DOI: 10.33480/pilar.v16i1.1293

[36] Wiguna W. Decision tree of Coronavirus Disease (COVID-19) surveillance. IEEE Dataport. 2020. DOI: 10.21227/remc-6d63