Newton Trees

Fernando Martínez-Plumed, Vicent Estruch, Cèsar Ferri, José Hernández-Orallo, and María José Ramírez-Quintana

DSIC, Universitat Politècnica de València, Camí de Vera s/n, 46022 València, Spain.
{fmartinez,vestruch,cferri,jorallo,mramirez}@dsic.upv.es

Abstract. This paper presents Newton trees, a redefinition of probability estimation trees (PET) based on a stochastic understanding of decision trees that follows the principle of attraction (relating mass and distance through the Inverse Square Law). The structure, application and, very especially, the graphical representation of these Newton trees provide a way to make their stochastically driven predictions compatible with user's intelligibility, so preserving one of the most desirable features of decision trees, comprehensibility. Unlike almost all existing decision tree learning methods, which use different kinds of partitions depending on the attribute datatype, the construction of prototypes and the derivation of probabilities from distances are identical for every datatype (nominal and numerical, but also structured). In this way, Newton trees can also be seen as a family of methods, since they can be parametrised by using any set of distances in the literature. Although we present a way of graphically representing the original stochastic probability estimation trees using a user-friendly gravitation simile, Newton trees can also be converted into crisp decision trees by deriving cutpoints. We include experiments showing that Newton trees outperform other PETs in probability estimation and accuracy.

Keywords: Probability Estimation Trees, Decision Trees, Distance Methods, Inverse Square Law, Stochastic Decision Trees.

1 Introduction

Decision tree learning [23] is one of the most popular (and powerful) techniques in machine learning and, very especially, in data mining. Two of the most important features of decision trees are their divide-and-conquer covering of the problem space and the use of decisions defined over univariate conditions (although multivariate variants exist). Decision tree learning has evolved through the introduction of datatype-specific condition schemes, dozens of splitting criteria, and many class assignment, pruning and stopping rules.

Probability Estimation Trees (PETs) [21][9], whose output is a probability rather than a crisp decision, are heirs of this technology, and are generally preferable over classical decision trees, whenever the goal is good rankings or good probability estimation. Initially, PETs were improved by using smoothing in the leaves [21] or through a pruning-smoothing [9]. The decision tree was unaltered and the rules which were derived from it were consistent with its predictions. However, many other recent extensions of PETs use the decision tree as a skeleton upon which a complex decision making process takes place. The way the decision tree looks and the way it must be used to obtain the predictions are no longer easy to understand or even consistent. So, these modern PETs have no real advantage over tree ensembles, since, if the prediction for an instance cannot be easily understood by looking at the decision tree, we lose the greatest advantage of decision trees, i.e. intelligibility, and we can use boosted, bagged or randomised tree ensembles instead, or even any other powerful black-box technique, such as SVM or neural networks.

Many PET innovations have evolved in such a way that the resulting technique is not intelligible any more. One explanation for this is that the old condition patterns found in classical decision tree learning ($X < value$ for numerical values, $X = value$ for nominal values, as in CART [4] or C4.5 [22]), have been preserved when constructing the tree, but understood in a different way when using the trees [18] [1] [2]. In other cases, new condition patterns (e.g. fuzzy, [26]) are used, but the user is generally not familiar with the linguistic concepts which are obtained in the fuzzyfication.

In an effort of getting the most from decision tree learning for probability estimation, in this paper we present a new Stochastic Probability Estimation Tree learning technique. Splits are constructed by using attribute prototypes which work as attractors, following an inverse square law using the distance to the prototype and its mass, similar to other 'gravitational' approaches in machine learning [28][6][15][12][20]. We will present the details of Newton trees in the following sections and we will show that they introduce a series of new features and important contributions, namely:

- We use the notion of distance in a univariate way as a general way of treating any kind of datatype (numerical, nominal, ordinal or structured).
- We construct the tree based on the principle of attraction and we derive the probabilities, use and represent the tree using the same principle.
- We handle numerical, nominal and ordinal attributes in the same way. We do not have to *type* the attributes but just provide a distance for each datatype. For numerical attributes it is frequently the absolute difference and for nominal/ordinal attributes it is the identity function or any specific cost function.
- We use mediods (prototypes from the set of attribute values) and not centroids, so properly handling both continuous and discrete datatypes. For continuous datatypes we only construct a cluster per attribute and class, and not a cutpoint between each pair of values. So, we have an order of $O(m)$ instead of $O(n \cdot m)$ partitions to evaluate, where m is the number of attributes and n is the number of examples.
- We provide a graphical representation of the trees to ease their interpretation.
- We evaluate the trees using one measure from each of the three most important families of measures to evaluate classifiers: accuracy, as a qualitative measure of error, AUC (*Area Under the ROC Curve*) as a measure of ranking quality, and MSE (*Mean Squared Error*) as a measure of calibration

and refinement quality, and we show they are significantly better than other PETs.
- Our trees are univariate but their partitions are not necessarily axis-parallel. We see that the partitions can create more expressive boundaries than classical decision trees without going multivariate.
- We provide the option to convert our stochastic trees into crisp trees, which can be written in the form of rules as in traditional decision trees.

The paper is organised as follows. The following section introduces notation and basic terminology on decision tree learning and probability estimation trees, by also reviewing some related work. Section 3 introduces Newton Trees, by first describing thbe attraction function and then how trees are learned and used to obtain the probability estimations. It also introduces a user-friendly representation of Newton trees based on the idea of attraction and gravitation. Section 4 includes a comprehensive bunch of experiments, which compare Newton Trees with a common PET (C4.5 without pruning and Laplace estimation). Section 5 shows the extended non-axis-parallel expressiveness of Newton Trees and two crisp derivations, by transforming the stochastic understanding of the attraction rule into a crisp cutpoint rule. Finally, section 6 closes the paper with the conclusions and the future work.

2 Notation and Previous Work

Decision tree learning [23] was introduced by Breiman et al. [4], and later on Quinlan developed some of the most well-known systems for learning decision trees, such as C4.5 [22]. Most algorithms that generate decision trees to classify instances employ a greedy top-down search. From a set of possible partitions or splits, a *splitting criterion* is used to select the optimal one, according to a given statistical or informational criterion. Each partition generates two or more descendants. This process is repeated for each new descendant node.

2.1 Notation

The set of all possible unlabelled examples E is composed of all the elements $e = \langle e_1, e_2, ..., e_m \rangle$ with m being the number of attributes. The attribute names are denoted by $\langle x_1, x_2, ..., x_m \rangle$. A labelled dataset D is a set of pairs $\langle e, i \rangle$ where $e \in E$ and $i \in C$, where C is the set of classes. The number of classes, $|C|$, is denoted by c. We define a probability estimator as a set of c functions $p_{i \in C} : E \to \mathcal{R}$ such that $\forall i \in C, e \in E : 0 \leq p_i(e) \leq 1$ and $\forall e \in E : \sum p_{i \in C}(e) = 1$. Decision trees are formed of nodes, splits and conditions. A *condition* is any Boolean function $g : E \to \{true, false\}$. A *split* or *partition* is a set of s conditions $g_k : 1 \leq k \leq s$. A *decision tree* can be defined recursively as follows: (i) a node with no associated split is a decision tree, called a leaf; (ii) a node with an associated split $g_k : 1 \leq k \leq s$ and a set of s children t_k, such that each condition is associated with one and only one child, and each child t_k is a decision tree, is also a decision tree. Given a node ν, $Children(\nu)$ denotes the set of its children and $Parent(\nu)$ denotes its predecessor node. The special node where $Parent(\nu) = \emptyset$ is called the *root* of the tree. After the training stage, the examples will have been distributed among all the nodes in the tree, where the root node contains all the examples and downward nodes contain the subset of examples that are consistent with all its ancestors' conditions. Therefore, every node has particular absolute frequencies $n_1, n_2, ..., n_c$ for each class. The cardinality of the node is given by $\sum n_i$. A *decision tree classifier* is defined as a decision tree with an associated labelling of the leaves with classes. Usually, the assigned class is the most frequent class in the leaf. A *probability estimation tree (PET)* is a decision tree which outputs a probability for each class. One typical way of generating these probability estimates is to use relative frequencies in the leaves $p_i = n_i / \sum(n_i)$ or some smoothing (Laplace, m-estimate) of them.

2.2 Related Work

Existing Probability Estimation Trees output a probability but are not necessarily probabilistic in nature. A first issue is that they typically use a divide-and-conquer philosophy for constructing the tree but the same philosophy is used to make a prediction. Given an example, a sequence of decisions will lead to a leaf of the tree where a value is returned (a class in classification trees, a number in regression trees, a probability in PETs, etc.). The rest of the information of the tree is wasted. In decision theory, though, this crisp view of decisions is awkward, since each decision can have an associated probability, and the overall probability must be computed by considering the whole structure of the tree. This kind of trees are frequently (but not always) called stochastic decision trees (e.g. [13]). In decision tree learning, the stochastic use of a decision tree is less common that the use of *several* trees. Option trees [5][17] and shared multitrees [11] are extensions of decision tree learning to consider alternative partitions at each point in the tree, but in the end it is not a single tree which is learned, but several, as in tree ensembles. More closely related to the notion of a stochastic decision tree is a probabilistic or Bayesian aggregation to decide whether and how a decision sequence has to be pursued or pruned (e.g. [5], [9]). However, to our knowledge, only [18] used all the information of a tree in order to get better probability estimates, apart from some fuzzy decision trees (e.g. [26]).

A second issue is that this use of all the paths in the tree, can be made in such a way that the probabilities of the tree are independent to the instance which is being processed. In fact, this has been the approach in [18], by using an ad-hoc parameter which is used to determine the probability of each child in a partition. More recent approaches, [1] [2] have made the probability depend on the proximity to the cut-point for the attribute, using Kernel Density Estimates. For instance, given a condition $X > 3$, it is assumed that the probability of pursuing that branch must be greater the highest the separation is to the cutpoint 3. However, these approaches still construct the tree in the classical way, and may disregard the cardinalities (i.e. masses) when using the tree to derive the probabilities. In other words, a tree can be constructed by a classical algorithm (such as C4.5 [22] or CART [4]) and its probabilistic or stochastic interpretation can be inconsistent to the way the decision tree was constructed. For instance, [2] use a way to estimate probabilities which is completely different to the way the tree is interpreted by humans, which is still done in a crisp way. In fact, this is what happens with many of the multivariate extensions of decision tree learning. They are limited to the leaves to preserve the intelligibility and the essence of a decision tree. For instance, [16] uses Naive Bayes at each leaf, but only in the leaves and not in the construction or in the internal nodes. In [29], this Naive Bayes philosophy is extended through the whole tree, which makes it difficult to extract knowledge from the trees, since it becomes a hierarchical Naive Bayes rather than a decision tree.

A third issue is how different datatypes are handled. Many of the previous approaches only deal with numerical attributes ([1], [2]) or only deal with nominal attributes. When handling both, the trees just preserve the very specific way of handling numerical attributes with cutpoints and nominal attributes with equalities, as C4.5 [22] or CART [4], which may have very refined (and sometimes ad-hoc) methods to derive the cutpoints. In fact, for numerical attributes, this takes most of the time complexity of the algorithm. Other classical decision trees, such as QUEST and CHAID [23] repeat this scheme: while ANOVA F-tests or Levene's tests are used for ordinal and continuous attributes, Chi-squared tests are used for numerical attributes. Even in the case of fuzzy decision trees (which in some cases provide a more integrated view of nominal and numerical attributes) it is unclear how decision trees can be applied to problems where some attributes are from other datatypes such as intervals, sequences, sets or other kind of structured data.

Having all the previous approaches to PETs, in this work we propose a new decision tree learning method which has been designed from scratch with the goals of being stochastic in nature, general and flexible in the way it handles data attributes, and intelligible. Our approach can be seen as a hierarchical or divide-and-conquer centre splitting method [24][8] where the distance function and mass is converted into a probability density using an inverse square law.

3 Stochastic Distance-based Probability Estimation Trees

In this section we define our Stochastic Probability Estimation Tree learning technique which leads to Newton trees.

3.1 Gravitational Partitions

When constructing splits, decision trees typically generate conditions which are then evaluated to see how well they separate the classes. Instead of that, we propose to define a node/cluster per class and then try to find the characterisation of each node in terms of one attribute at a time (univariate).

Following this idea, one first approach is to use Kernel Density Estimation [25] in order to derive a probability density function (pdf), from the examples belonging to each class. However, many of these techniques will construct a parametrised or composite pdf that will make partitions unintelligible, apart from having the risk of overfitting. Another (related) approach is to derive a prototype for each node, and then, to derive a probability from the prototypes. In order to treat discrete datatypes appropriately, we use a mediod (the element in each cluster which its average distance to the rest is the lowest). If we generate prototypes, one possibility to derive probabilities from them is to assume some probability distribution. For instance, if we consider a normal distribution for each node with centre at the prototype and with standard deviation equal to the mean of distances of the elements of the node, we have a pdf. Figure 1 (left) shows the pdf using a Gaussian with centres 3 and 8, with standard deviations 1 and 3.5 (respectively) and masses 20 and 100 (respectively). This can be converted into probabilities by mere normalisation, as shown in Figure 1 (right).

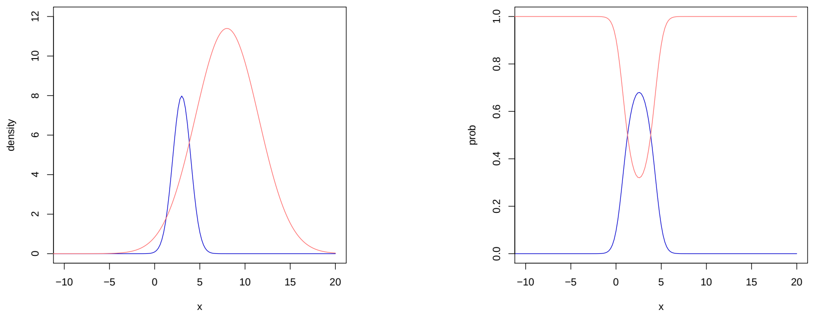


Fig. 1. Two normal distributions placed at centres 3 and 8, with standard deviations 1 and 3.5 (respectively) and masses 20 and 100 (respectively)(left). The probabilities derived from the Gaussians (right).

The problem of the previous approach is that when masses are too despair, one distribution can cover the other, giving a plain (and useless) partition where all the elements go to one prototype. One criterion to avoid this is to give extra importance to the distance in such a way that at distance 0 the probability is always 1. A simple way to do this is to employ an inverse-square law such as in gravitation. Using this, we define the following *attraction* function between an element e of mass m_e (we will assume $m_e = 1$) and a prototype π of mass m_π separated by a distance $d(e, \pi) = d$:

$$attraction(e, \pi) = \frac{m_e m_\pi}{d(e, \pi)^2} = \frac{m_\pi}{d(e, \pi)^2}$$

We are interested in deriving class probabilities considering this attraction. Figure 2 shows the attraction (left) and the probability (right) with the same parameters as before (note that the standard deviation is no longer used).

An interesting property is that when the distance goes to infinity the probabilities tend to converge to the mass proportion. For instance, if we have two centres at 3 and 8, and 8 has much more mass (as in the previous example), it seems more logical to expect that the attraction to 8 will be higher than the attraction to 3 for a point placed at -100.

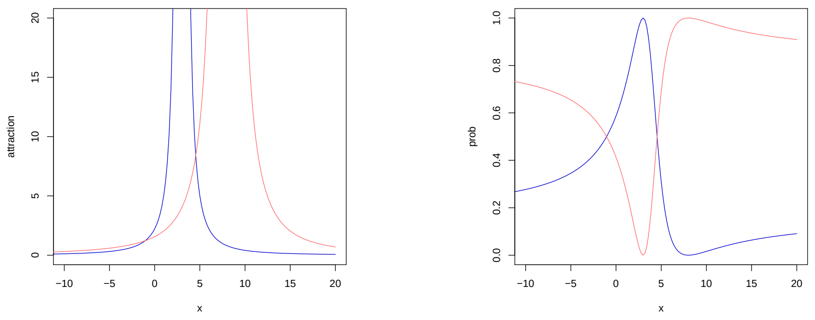


Fig. 2. Two gravitational centres at 3 and 8 with masses 20 and 100 (respectively)(left). The probabilities derived from the gravitational centres (right).

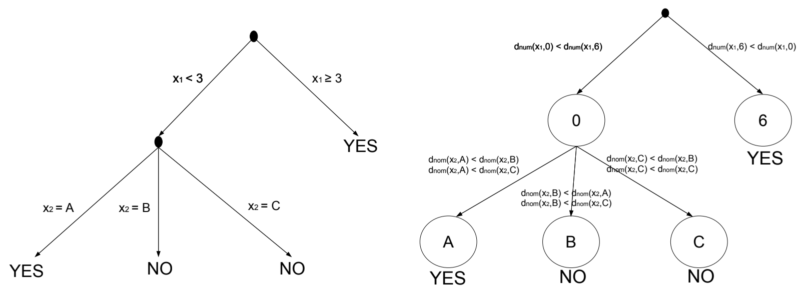
Of course, the idea of using the gravitational law in machine learning is not new at all, for instance in clustering ([28], [12]), in visualisation ([6]) or classification ([20]). In fact, the same Inverse Square Law principle is present in some variants of Kernel Density Estimation, several classification techniques such as weighted kNN, where the weight is a kernel which is simply defined as the inverse of the distance, or in some other clustering algorithms. To our knowledge, its use for decision trees is new.

3.2 Tree Generation

We based our method on the use of *prototypes* and distances to define partitions in a hierarchical way, which resembles the centre splitting method [24]. Basically, the centre splitting method consists in dividing the input space in different regions where each region is represented by a centre (which may match to an existing example or not). In every iteration of the process, a centre is calculated for every different class which is presented in the area. Then, every example is associated to its nearest centre. This process is repeated until the area is pure. One of the special features of this method is that the examples are managed as a whole, which precludes the use of univariate partitions. This appreciation leads us to propose a decision tree inference strategy where partitions are made only taking into account one attribute at a time. In this way centroids are computed considering only the values of one attribute, which allows us to join centre splitting and decision tree learning techniques in an elegant way.

The detailed definition of the algorithm can be found in [19]. Here, we give a more sketchy description. Basically, the tree generation algorithm works as follows: for each attribute x_r and for each class i, a prototype $\pi_{r,i}$ is calculated as the attribute value with lowest mean distance to the elements of the class. Once this process is finished, the splitting attribute is selected according to one of the well-known splitting criteria (for instance, gain ratio [22]). Then, the split proceeds by associating every instance to its closest attribute prototype, which typically produces impure clusters. Note that, during the splitting process, we apply the *attraction* function assuming that the mass is the unit. This is due to the fact that the total mass of a node is not known until all the instances have been associated to its prototype. Under that assumption, the process becomes simpler and more efficient. Although the computation of distances is quadratic on the number of instances, we can reduce by using a distance matrix per attribute (of size $n_r \times n_r$, where n_r are the number of different attribute values) prior to the algorithm execution. For numerical attributes we can just compute a mean if the absolute difference distance is used. But, more importantly, if we have m attributes and n_r values per attribute, we only construct (and evaluate) $O(m)$ partitions and not $O(n_r \times m)$, the typical order for classical decision tree learning algorithms using midpoints for continuous attributes. Additionally, it is important to note that distances are computed between attribute values and not between whole examples. This issue is also crucial for efficiency.

Figure 3 shows an example of a traditional decision tree and an equivalent distance-based tree constructed for a problem with a numerical attribute (x_1), a nominal attribute (x_2) and two classes (YES and NO). The absolute difference distance is denoted by d_{num} and the 0/1 discrete distance is denoted by d_{nom}.



$x_1 < 3$
$x_1 \geq 3$
YES
$x_2 = A$
$x_2 = B$
$x_2 = C$
YES
NO
NO
$d_{num}(x_1,0) < d_{num}(x_1,6)$
$d_{num}(x_1,6) < d_{num}(x_1,0)$
0
6
YES
$d_{nom}(x_2,A) < d_{nom}(x_2,B)$
$d_{nom}(x_2,A) < d_{nom}(x_2,C)$
$d_{nom}(x_2,B) < d_{nom}(x_2,A)$
$d_{nom}(x_2,B) < d_{nom}(x_2,C)$
$d_{nom}(x_2,C) < d_{nom}(x_2,B)$
$d_{nom}(x_2,C) < d_{nom}(x_2,C)$
A
B
C
YES
NO
NO

Fig. 3. A traditional (Left) and distance-based (Right) decision tree.

3.3 Stochastic Probability Calculation

Now, we illustrate how a Newton Tree is used to estimate probabilities in a stochastic way. In what follows, $\overrightarrow{p}(\nu, e) = \langle p_1(\nu, e), \ldots, p_c(\nu, e)\rangle$ denotes the probability vector of example e at node ν, where $p_i(\nu, e)$ denotes the probability that e belongs to class i at node ν. With $\hat{p}(\nu, e)$ we denote the probability that e falls into node ν (coming from its parent), which is derived from the attraction that ν exert over e, that is

$$\hat{p}(\nu, e) = \frac{attraction(e, \nu)}{\sum_{\mu \in Children(Parent(\nu))} attraction(e, \mu)}$$

Given a new example e and a Newton tree T, the objective is to calculate the probability vector at the root of T, $\overrightarrow{p}(root, e)$. Basically, the idea is to compute downwards the probability of falling in each leaf, calculate the leaf probability vector and then to propagate upwards the leaf probability vector to the root to

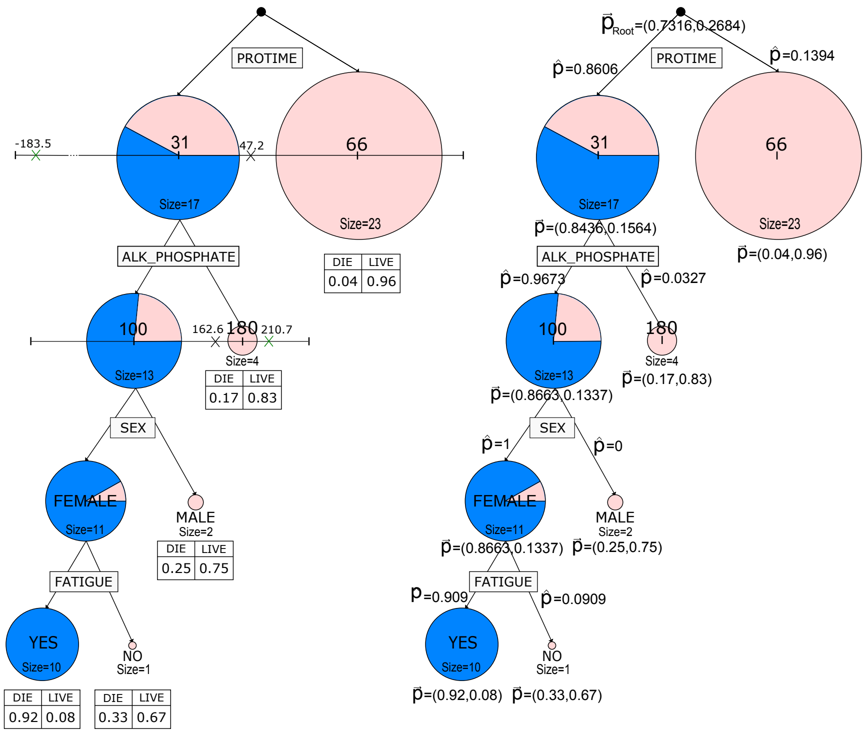


Fig. 4. (Left) Newton Tree for the hepatitis dataset. (Right) The node probability vectors, children probabilities and global probability vector for example (PROTIME=40, ALK_PH=120, SEX=FEMALE, FATIGUE= UNKNOWN)

obtain the total class probability vector $\overrightarrow{p}(root, e)$. The leaf probability vectors can be obtained once the tree T has been built by applying Laplace as has been shown in [21, 9]. For each example, we calculate the probability of choosing each child node μ if placed at the parent node ν using the attraction (i.e., $\hat{p}(\mu, e)$. This probability is multiplied by the probability vector of the child ($\overrightarrow{p}(\mu, e)$):

Definition 1. Stochastic Probability Vector Estimation
Given an example e and a Newton tree T, the probability vector $\overrightarrow{p}(root, e)$ at the root of T is estimated by applying

$$\forall \nu \in T : \overrightarrow{p}(\nu, e) = \begin{cases} \sum_{\mu \in Children(\nu)} \hat{p}(\mu, e) \cdot \overrightarrow{p}(\mu, e) & \textit{if } \nu \textit{ is not a leaf} \\ \langle Laplace(1, \nu), \ldots, Laplace(c, \nu) \rangle & \textit{if } \nu \textit{ is a leaf} \end{cases}$$

where $Laplace(j, \nu)$ is the Laplace correction of the frequency of elements of class j in node ν.

The stochastic calculation of the probabilities seen above may seem too cryptic for a general use of these trees if intelligibility is a requirement. If we take a look at the representation of trees in Figure 3 (right), comprehensibility is highly at stake. In order to address this issue, we show a graphical representation of Newton trees, which may help users understand how the stochastic probability assignment is made, and to get insight from the tree.

Figure 4 (left) shows this user-friendly representation of a Newton Tree for the Hepatitis dataset from the UCI repository [3]. Note that all partitions are binary because this is a two-class problem, namely DIE and $LIVE$. The two first splits are made over the numerical attributes $PROTIME$ and $ALK\_PHOSPHATE$, respectively, and the other two splits are made over the nominal attributes SEX and $FATIGUE$. The nodes of the tree are represented as balls of a size which is proportional to the node mass. For instance, the left-hand side node at the first level of the tree has a mass of 17, which means that 17 training examples fall into this node. The ball also shows in different colour the proportion of examples of each class. Additionally, since each node is referred by its prototype, the value for the attribute prototype is shown in the middle of each ball. For instance, the value of attribute $PROTIME$ is 31 on the left prototype at the first level of the tree. Finally, the smoothed probabilities per class at the leaves are also provided (in the figure, as a small table below each leaf). In order to ease the understanding on how probabilities are derived, Figure 4 (right) shows the internal probabilities (vectors and node probabilities) and the top vector probability for example $(PROTIME = 40; ALK\_$
$PHOSPHATE = 120; SEX = FEMALE; FATIGUE = UNKNOWN)$, which is $(0.7316, 0.2684)$, a relatively clear DIE case. All these graphical elements we have included in the Newton Trees representation may help a possible final user understand the way in that probabilities are estimated, making Newton trees less cryptic than other probability estimation tree methods, especially those which are stochastic.

4 Experiments

The aim of this section is to compare Newton trees with a common implementation of Probability Estimation Trees, namely unpruned decision trees with Laplace smoothing in the leaves as suggested by [21][9]. In particular, we chose J48 (the variant of C45.) implemented in Weka [27]. We used *Gain ratio* as splitting criterion for Newton trees and J48. The evaluation has been performed over 30 datasets from the UCI repository [3], from which we removed instances with missing values and classes without examples (see Table 1 for their characteristics). We set up a 20×5-fold cross validation, making a total of 100 learning runs for each pair of dataset and method (3,000 overall). As evaluation metrics we used the three most important families of measures to evaluate classifiers as defined in [10]: accuracy, as a qualitative measure of error, AUC (*Area Under the Curve*) as a measure of ranking quality, and MSE (*Mean Squared Error*) as a measure of calibration and refinement quality.

Table 2 shows the average accuracy, AUC and MSE obtained by the two algorithms. At the bottom, we also show the mean values for all the datasets. These means are just illustrative. To analyse whether the differences are signifi-

DataSet	Classes	Nom.	Num.	Size	Majority Class	Minority Class
Anneal	6	32	6	898	(3)684	(4)
Autos	7	10	15	159	(0)48	(-3)0
autos_5c	5	10	15	156	(0)48	(3)13
Balance-Scale	3	0	4	625	(L),(R)288	(B)49
Breast-Cancer	2	9	0	277	(non-recurrence)196	(recurrence)81
chess-kr-vs-kp	2	36	0	3196	(won)1669	(nowin)1527
cmc	3	7	2	1473	(1)629	(2)333
Credit-a	2	9	6	653	(-)357	(+)296
Credit-g	2	13	7	1000	(good)700	(bad)300
Diabetes	2	0	9	768	(positive)500	(negative)268
Glass	7	0	9	214	(build wind non-float)76	(vehic wind non-float)0
Heart-statlog	2	0	13	270	(absent)150	(present)120
Hepatitis	2	14	5	80	(live)67	(die)13
Ionosphere	2	0	34	351	(g)225	(b)126
Iris	3	0	4	150	50	50
Monks 1	2	6	0	556	(0)278	(1)278
Monks 2	2	6	0	601	(0)395	(1)206
Monks 3	2	6	0	554	(1)288	(0)266
Mushrooms	2	22	0	5644	(e)3488	(p)2156
new-thyroid	3	0	5	215	(1)150	(3)30
pimaW	2	0	8	768	(0)500	(1)268
Sonar	2	0	60	208	(mine)111	(Rock)97
SoyBean	19	36	0	562	(brown-spot)92	(herbicide-injury)0
spectf_train	2	0	44	80	(0)40	(1)40
Tae	3	2	3	151	(3)52	(1)49
Tic-TacW	3	8	0	958	(1)626	(0)332
Vehicle3C	3	0	18	846	(saab_bus)435	(van)199
Vote	2	16	0	435	(democrat)267	(republican)168
Vowel	11	3	10	990	90	90
Wine	3	0	13	178	(2)71	(3) 48
Zoo	7	16	1	101	(mammal)41	(reptile)5

Table 1. Description of the datasets used in experiments.

cant, we used the Wilcoxon signed-ranks test with a confidence level of $\alpha = 0.05$ and $N = 30$ data sets, as suggested in [7]. Significant differences are shown in bold. Finally, in Table 3 we focus on these differences, showing an entry $w/t/l$ for each measure and dataset subset, which indicates that Newton trees win in w, tie in t, and lose in l datasets, compared to the J48 PETs.

From the tables, we see that Newton trees outperform J48 PETs in the three measures (Accuracy, AUC and MSE), and with the means in Table 2, in any selection depending on the type of dataset (multiclass/binary, nominal/numerical/mixed). The strongest differences are found in AUC, which is the recommended measure when evaluating PETs ([14]). If we look at the significance results in Table 3, we have a similar picture. The exception is the result for nominal datasets. While AUC is still much better, the results in MSE are worse (and as a result so is accuracy). This indicates a bad calibration of the results for datasets with only nominal partitions, which might be caused by the way discrete distances affect on the attraction measure, although more research should be done to clarify this (since there are only 7 datasets in this subset).

5 Expressiveness and Non-stochastic Versions

Newton tree construction agrees with classical decision tree learners in that both are based on univariate partitions, as we have shown in Section 3. In the case of traditional algorithms, this means that partitions are axis-parallel and divide

Name	Classes	Att Type	Newton Trees Acc.	Newton Trees AUC	Newton Trees MSE	Unpruned Laplace J48 Acc.	Unpruned Laplace J48 AUC	Unpruned Laplace J48 MSE
anneal	6	Mixed	97.5110	**0.8943**	0.0119	**98.7800**	0.8890	**0.0073**
autos_5c	5	Mixed.	**79.5060**	**0.9043**	0.0825	77.7130	0.8827	0.0840
balance-scale	3	Num.	**79.5520**	0.7962	0.1050	78.6880	**0.8199**	**0.0998**
breast-cancer	2	Nom.	**73.0110**	**0.6436**	**0.1929**	67.9360	0.6084	0.2233
chess-kr-vs-kp	2	Nom.	98.5050	0.9975	0.0135	**99.3050**	**0.9988**	**0.0064**
cmc	3	Mixed.	**50.1720**	**0.6739**	**0.2025**	49.1100	0.6658	0.2107
credit-a	2	Mixed.	**84.9310**	**0.9107**	**0.1118**	82.7960	0.8982	0.1256
credit-g	2	Mixed.	**70.3300**	**0.7202**	**0.1897**	68.2900	0.7016	0.2159
diabetes	2	Num.	71.8630	0.7801	**0.1798**	**72.8070**	0.7772	0.1877
glass	7	Num.	67.2940	0.7828	0.0901	67.0340	0.7895	0.0879
heart-statlog	2	Num.	**78.0740**	**0.8626**	**0.1490**	76.1850	0.8398	0.1753
hepatitis	2	Mixed.	**83.4370**	**0.7570**	**0.1143**	79.4370	0.6542	0.1498
ionosphere	2	Num.	88.9160	0.9235	0.0916	88.8460	0.9195	0.0917
iris	3	Num.	**94.7660**	**0.9938**	**0.0315**	94.0330	0.9710	0.0349
monks1W	2	Nom.	93.5230	**0.9899**	0.0606	92.7690	0.9761	**0.0519**
monks2W	2	Nom.	**85.8750**	**0.9378**	**0.1124**	61.3790	0.6456	0.2348
monks3W	2	Nom.	98.6730	**0.9926**	0.0166	98.6370	0.9909	**0.0135**
mushroom	2	Nom.	99.9910	0.9999	0.0193	100.0000	1.0000	**0.0001**
new-thyroid	3	Num.	92.6970	**0.9854**	**0.0438**	92.3480	0.9237	0.0454
pimaW	2	Num.	71.8630	0.7801	**0.1798**	**72.7750**	0.7772	0.1877
sonar	2	Num.	**77.5990**	**0.8499**	**0.1538**	73.3710	0.7888	0.2162
soybean	19	Nom.	89.2420	0.9771	0.0228	**91.2270**	0.9770	**0.0183**
spectf_train	2	Num.	67.3120	0.7301	0.2097	**71.7500**	0.7365	0.2196
tae	3	Mixed.	**58.7010**	**0.7398**	**0.1877**	54.1660	0.7078	0.1996
tic-tacW	3	Nom.	78.1110	0.8526	0.1426	**79.3990**	**0.8699**	**0.1393**
vehicle3c	3	Num.	72.1210	0.8441	0.1355	**73.0240**	**0.8807**	**0.1251**
vote	2	Nom.	94.5020	**0.9892**	0.0383	**95.1370**	0.9827	**0.0355**
vowel	11	Mixed.	75.3580	**0.9671**	0.0578	**79.5400**	0.9157	**0.0447**
wine	3	Num.	**94.3840**	**0.9905**	**0.0408**	92.2070	0.9544	0.0471
zoo	7	Mixed.	**94.9020**	**0.7243**	0.0252	93.1610	0.7147	**0.0234**
Mean (All)			82,0907	0,8664	0,1004	80,7283	0,8419	0,1101
Mean $(c = 2)$			83,6503	0,8665	0,1146	81,3388	0,8310	0,1334
Mean $(c > 2)$			80,3084	0,8662	0,0843	80,0307	0,8544	0,0834
Mean (Nominal)			90,1592	0,9311	0,0688	87,3099	0,8944	0,0803
Mean (Numerical)			79,7034	0,8599	0,1175	79,4223	0,8482	0,1265
Mean (Mixed)			77,2053	0,8102	0,1093	75,8881	0,7811	0,1179

Table 2. Comparison between Newton trees and unpruned J48 with Laplace correction.

Unpruned Laplace J48 / Netwon Trees	Acc.	AUC	MSE
All	14/6/10	18/8/4	14/4/12
Nominal	2/3/4	5/2/2	2/0/7
Numerical	5/3/4	5/5/2	7/3/2
Mixed	7/0/2	9/0/0	5/1/3

Table 3. Aggregated results using the statistical tests

the space into rectangular areas that determine the points of the input space that belong to each class. However, Newton tree partitions are *not* axis-parallel. This is due to the fact that class boundaries are defined stochastically, and information from the whole tree is used. Figure 5 (Left) shows the class boundaries for the Newton tree built for the Hepatitis dataset (only considering the two upper levels). Axes represent the attributes used for the two first partitions (as we have seen in Figure 4). Class DIE is represented in (dark) blue and class $LIVE$ in (light) pink. Near each axis, there are two little balls that represent the prototypes (of the colour of its majority class), placed at their corresponding values.This capability of constructing curved boundaries may explain part of the increase of performance wrt. traditional PETs, even more when we in fact evaluate a much lower number of partitions per node.

We argued in Section 3 that Figure 4 allows users to get insight from the tree and explain its predictions. Nevertheless, it seems reasonable to see whether we can use Newton trees in a non-stochastic way and derive crisp partitions from it. For nominal attributes and using the discrete distance the transformation is trivial:

Definition 2. *Let x_i be a nominal attribute, and let $\pi_{i,k}, 1 \leq k \leq s$ be the computed prototypes in a node. Then, this split is defined in a crisp way as the set of s conditions $x_i = \pi_{i,k}$. If there are more attribute values than prototypes, then we choose the most populated node and mark it as the "all the rest" node.*

For numerical attributes, the idea is to calculate cutpoints:

Proposition 1. *Given a numerical attribute x_i using the absolute difference distance and given two consecutive prototypes $\pi_{i,j}$ and $\pi_{i,k}$, with $\pi_{i,j} < \pi_{i,k}$, the values for the numeric attribute x_i defined as*

$$t_1 = a + sqrt\left(\frac{a^2}{4} - \frac{b}{c}\right) \qquad t_2 = a - sqrt\left(\frac{a^2}{4} - \frac{b}{c}\right)$$

where $a = (m_{\pi_{i,k}} \cdot \pi_{i,j} - m_{\pi_{i,j}} \cdot \pi_{i,k})/c$, $b = (m_{\pi_{i,j}} \cdot \pi_{i,k}^2 - (m_{\pi_{i,k}} \cdot \pi_{i,j}^2)$ and $c = (m_{\pi_{i,j}} - m_{\pi_{i,k}})$, satisfy that, if $t_1 < t_2$[1]*,*

$$\forall e : \begin{cases} \textit{if } e_i < t_1 & \textit{then } \; e \textit{ falls into prototype } \pi_{i,k} \\ \textit{if } t_1 < e_i \leq t_2 & \textit{then } \; e \textit{ falls into prototype } \pi_{i,j} \\ \textit{if } t_2 \leq e_i & \textit{then } \; e \textit{ falls into prototype } \pi_{i,k} \end{cases}$$

Proof. Since prototypes in Newton trees exert an attraction force over examples, a given example e will fall into the prototype with the highest probability. As we have seen in Section 3.1, probability densities are derived from attraction forces. Given two prototypes and their probability densities, the points where densities cut each other determine the interval in that each prototype prevails. Those points are obtained by equalling the two attraction functions:

$$\frac{m_{\pi_{i,j}}}{(t - \pi_{i,j})^2} = \frac{m_{\pi_{i,k}}}{(t - \pi_{i,k})^2}$$

Solving this quadratic equation for t we have two solutions: t_1 and t_2. □

Figure 4 (Left) shows the cutpoints for the two first partitions. Note that one of the cutpoints is always between the prototypes and the other is on the left or on the righ of one of the prototypes. According to Proposition 1, the first partition for this examplecan be expressed by the conditions:

```
if (-183 < PROTIME) and (PROTIME <= 47.2) then fall into prototype 31 (...)
                                          else fall into prototype 66 (class=LIVE)
```

[1] If $t_2 < t_1$ the proposition trivially holds by exchanging both values.

Now, crisp partitions are axis-parallel. Figure 5 (Center) presents the partitions corresponding to the two first splits for the Hepatitis dataset using the two cutpoints. We can observe that the outside cutpoint typically limits the classes in a very sparse region, whereas the cutpoint in between the prototypes defines the class boundary in a much denser region. This suggests that only using one cutpoint, as Figure 5 (Right) presents, we could obtain a simpler but still competitive crisp transformation.

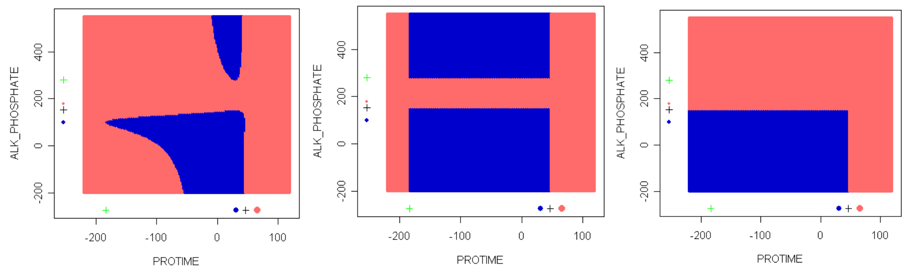


Fig. 5. Class boundaries for the Hepatitis dataset: original Newton tree (Left), Newton tree with two crisp cutpoints (Center) and with one crisp cutpoint (Right).

In order to analyse how much performance we lose between the original stochastic version and the crisp versions, we carried out an empirical evaluation of our three approaches to assess their performance: Crisp Newton Trees with one cutpoint, Crisp Newton Trees with two cutpoints and the original stochastic Newton trees. Table 4 summarises the average results obtained for the datasets used in the previous section. In order to simplify the process we restricted the number of children per node to 2. We see that the crisp approaches (especially the one with only one cutpoint) are competitive to the stochastic original version. However, the stochastic version is better in terms of AUC and especially in terms of MSE. This can also be related to the issue that we have restricted the number of children, and this forces the nominal partitions to have a catch-all (all-the-rest) by-default node.

	NCrisp 1			**NCrisp 2**			**Stochastic**		
	ACC	**AUC**	**MSE**	**ACC**	**AUC**	**MSE**	**ACC**	**AUC**	**MSE**
Mean (All)	81.7344	0.8545	0.1728	81.2794	0.8468	0.1754	81.8084	0.8671	0.1008
Mean. $(C = 2)$	82.9629	0.8592	0.2523	82.8376	0.8567	0.2544	83.6503	0.8665	0.1146
Mean. $(C > 2)$	80.3304	0.8492	0.0819	79.4986	0.8354	0.0852	79.7033	0.8677	0.0851
Mean (Nom.)	90.0140	0.9266	0.2864	90.4361	0.9284	0.2862	89.8192	0.9311	0.0690
Mean (Num.)	79.0648	0.8498	0.1295	77.9973	0.8367	0.1352	79.7468	0.8618	0.1170
Mean (Mixt.)	77.0143	0.7888	0.1170	76.4988	0.7784	0.1183	76.5462	0.8100	0.1110

Table 4. NCrisp 1 = Crisp Newton Trees with one cutpoint , NCrisp 2 = Crisp Newton Trees with two cutpoints, Stochastic = Stochastic Newton trees.

6 Conclusions and Future Work

This paper has presented a novel probability estimation tree learning method which is based on computing prototypes in the partitions and applying an Inverse Square Law that uses the distance to the prototype and its mass, in order to derive an attraction force which is then converted into a probability. The trees can be graphically represented in such a way that their meaning and patterns can be understood, and the predictions can be followed and explained, which is not easy in decision trees which are used stochastically. Even though Newton trees are still univariate, we have seen that they have the power to express and construct non-axis-parallel boundaries, so its pattern expressiveness is higher than traditional decision trees. Since Newton trees are relatively easy to implement, and we have presented a relatively user-friendly graphical represenation, we think that they could appear in machine learning toolkits and data mining suites in the future.

The use of prototypes (mediods) instead of centroids allows for the use of our trees for any kind of datatype (continuous or discrete), as long as we provide a distance function for each datatype. Consequently, we can apply our trees to structured datatypes, such as sequences, sets, ordinal data (which would not require a numerisation but a proper distance), intervals or even images and texts. More importantly, we can use the tree with a mixture of all these datatypes. If distance matrices are preprocessed (only once for each attribute before start), the computation of the prototypes is much more efficient than the split population schemes in traditional decision trees, since we group by classes and then compute the mediod of each cluster. Consequently, the number of different splits to evaluate at each node is equal to the number of attributes and does not depend on midpoints or the size of the dataset.

There are many research lines to pursue. One is to use the mass also when constructing the tree or using all the attribute values as possible clusters. However, these two modifications would entail extra computational cost and could only be justified if there is a significant improvement in the results.

The handling of null (missing) values is straightforward in our setting. Typically, since distance can be considered equal to every prototype, mass will be the only factor for the probability, which seems quite reasonable. We have not included null values in the experiments to make comparisons easier to follow, but a study on whether our trees are more robust (as we expect) to missing values (and also to outliers/noise) is an experiment we will have to perform in the near future. As a more ambitious future work, we plan the application/extension of Newton Trees to regression and clustering and other machine learning tasks.

References

1. I. Alvarez and S. Bernard. Ranking cases with decision trees: a geometric method that preserves intelligibility. In *IJCAI*, pages 635–640, 2005.
2. I. Alvarez, S. Bernard, and G. Deffuant. Keep the decision tree and estimate the class probabilities using its decision boundary. In *IJCAI*, pages 654–659, 2007.
3. C. Blake, E. Keogh, and C. Merz. UCI repository of machine learning databases, 1998. (http://www.ics.uci.edu/mlearn/MLRepository.html).
4. Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees.* Chapman & Hall, New York, NY, 1984.
5. W. Buntine. Learning classification trees. *Stats. and Computing*, 2(2):63–73, 1992.

6. Ian Davidson. Visualizing clustering results. In *SIAM Conf. on Data Mining*, 2002.
7. J. Demsar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, January 2006.
8. V. Estruch. Bridging the gap between distance and generalisation: Symbolic learning in metric spaces. PhD Thesis, DSIC-UPV http://www.dsic.upv.es/∼vestruch/thesis.pdf, 2008.
9. C. Ferri, P. Flach, and J. Hernandez-Orallo. Improving the auc of probabilistic estimation trees. In *Proc. ECML*, volume 2837 of *LNCS*, pages 121–132, 2003.
10. C. Ferri, J. Hernández-Orallo, and R. Modroiu. An experimental comparison of performance measures for classification. *Pattern Recogn. Lett.*, 30(1):27–38, 2009.
11. C. Ferri, J. Hernández-Orallo, and M. Ramírez-Quintana. From ensemble methods to comprehensible models. In *Discovery Science*, pages 223–234. Springer, 2002.
12. J. Gomez, D. Dasgupta, and O. Nasraoui. A new gravitational clustering algorithm. In *Int. Conf. on Data Mining*, page 83. Society for Industrial & Applied, 2003.
13. R.F. Hespos and P.A. Strassmann. Stochastic decision trees for the analysis of investment decisions. *Management Science*, 11(10):244–259, 1965.
14. Jin Huang and Charles X. Ling. Using auc and accuracy in evaluating learning algorithms - appendices. *IEEE Trans. Knowl. Data Eng.*, 17(3), 2005.
15. M. Indulska and ME Orlowska. Gravity based spatial clustering. In *Proc. Int. Sym. on Advances in geographic information systems*, page 130, 2002.
16. R. Kohavi. Scaling up the accuracy of naive-Bayes classifiers: A decision-tree hybrid. In *Int. Conf. on Knowledge Discovery and Data Mining*, volume 7, 1996.
17. R. Kohavi and C. Kunz. Option decision trees with majority votes. In *Int. Conf. on Machine Learning, ICML*, pages 161–169, 1997.
18. C.X. Ling and R.J. Yan. Decision tree with better ranking. In *International Conference on Machine Learning*, volume 20-2, page 480, 2003.
19. F. Martinez-Plumed, V. Estruch, C. Ferri, J. Hernández-Orallo, and M. J. Ramírez-Quintana. Newton trees. extended report. Technical report, DSIC, UPV, http://www.dsic.upv.es/∼flip/NewtonTR.pdf, 2010.
20. L. Peng, B. Yang, Y. Chen, and A. Abraham. Data gravitation based classification. *Information Sciences*, 179(6):809–819, 2009.
21. Foster J. Provost and Pedro Domingos. Tree induction for probability-based ranking. *Machine Learning*, 52(3):199–215, 2003.
22. J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA, 1993.
23. Lior Rokach and Oded Maimon. *Data Mining with Decision Trees: Theory and Applications.* World Scientific, 2008.
24. C.J. Thornton. *Truth from trash: how learning makes sense.* The MIT Press, 2000.
25. Berwin A. Turlach. Bandwidth selection in kernel density estimation: A review. In *CORE and Institut de Statistique*, 1993.
26. M. Umano, H. Okamoto, I. Hatono, H. Tamura, F. Kawachi, S. Umedzu, and J. Kinoshita. Fuzzy decision trees by fuzzy ID 3 algorithm and its application to diagnosis systems. In *3rd IEEE Conf. on Fuzzy Systems*, pages 2113–2118, 1994.
27. I.H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann Pub, 2005.
28. WE Wright. Gravitational clustering. *Pattern Recognition*, 9(3):151–166, 1977.
29. H. Zhang and J. Su. Learning probabilistic decision trees for AUC. *Pattern Recognition Letters*, 27(8):892–899, 2006.

A Algorithm

Before introducing the different algorithms we requires some notation. $C = c_1, \ldots, c_k$ is a set of k class labels. A (labeled) training example e = (x; y) is represented by a tuple of attribute values $x = (x1, \ldots, xn)$ and a class label $y \in C$. An unlabeled example is represented by its attributes, that is $e = (x1, \ldots, xn)$. $Attr_x j$ returns the $j - th$ attribute of example e. Similarly, $Class(e)$ returns the class of example e. Additionally, two important functions to reduce the complexity of computing the distances are needed: $Values(S, X_j)$ returns the number of different values for the attribute x_j and $Card(v, S, X_j)$ returns the number of occurrences of value v in the attribute x_j in the sample S, namely $|\{e\ inS : Attr_{x_j} = v\}|$. The function $distance(x, y)$ computes the distance between the values of attributes x and y, which must be of the same type. The $attraction(e, \pi)$ function define the attraction between an instance e and a prototype π such as explained in the seccion 3 . Finally we have two functions that work with the nodes of decision tree: $Children(\nu)$ returns a set of all successors of node ν and $Parent(\nu)$ returns the predecessor node of ν.

The algorithm 1 is the main procedure of our method. The inputs are a training dataset of the form $(x_1, \cdots, x_n), n \geq 1$, a parameter m which limits the maximum number of child nodes per division (if weset to $m = 2$ we only have binary divisions) and a metric space ms where distances are defined between different attributes of the set of examples.

Therefore, $TreeGeneration$ algorithm is just a typical decision tree learning algorithm which, in this case, determines, for each attribute, a ranked list of prototypes which will lead to a set of children for each node. The main difference with the classical decision trees learners lies in four functions: *Compute_Prototypes*, *Attracts*, *ProbTree* and *ProbClass*.

The function *Attracts* (see Algorithm 2) just determines which prototype is assigned with a new example. This algorithm can be implemented in multiple ways (eg, considering the density or not) but we have chosen the simplest: returns the prototype nearest to the sample. In case of a tie, it returns the rigthmost prototype.

The function *Compute_Prototypes* (see Algorithm 3) is the most important one. This function can be performed in many different ways. Below, we show one of these possibilities. which just select the best prototype (in the way that the distances to the prototype for the elements of the same class are minimised), removes its class and the value of the attribute and looks for the next best prototype for a different class and value for the attribute, and so on until the limit m (given by the user) or the number of classes or attribute values is reached. Once the list of prototypes has been calculated, each prototype is associated with the examples that attracts.

Note that, the above procedure is independent of the attribute types. In fact,it can be applied to any kind of attribute and not only to nominal and numerical attributes as happens in classical decision tree algorithms. Even more, all attributes are handled in a similar way. This does not occur in other decision tree learners in which the partitions of numerics are made quite differently from

Algorithm 1 $TreeGeneration(S, m, ms)$

```
Require: $S$ is a set of examples of the form: $(x_1, \cdots, x_n), n \geq 1$, $m$ is the maximum number of children per node, $ms$ is the metric space.
 1: $Heuristics \leftarrow 0$
 2: $ProtList \leftarrow 0$
 3: if $S = 0$ then
 4:     return
 5: end if
 6: for all attribute $x_j$ do
 7:     $Heuristics_{x_j} \leftarrow Optimality(S, x_j, ms)$ //Gain ratio, GINI, etc.
 8: end for
 9: $m_{x_j} \leftarrow Argmax_{x_j}(Heuristics)$
10: if $Heuristics[m_{x_j}] = 0$ then
11:     return Leaf
12: else
13:     $ProtList \leftarrow ComputePrototypes(m_{x_j}, S, m, ms, C)$
14:     for $i = 1$ to $length(ProtList)$ do
15:         $TreeGeneration(ProtList[i], m, ms)$
16:     end for
17: end if
```

Algorithm 2 $Attracts(e, ProtList, x_j)$

```
Require: $e$ an example, $ProtList$ a ranked list of prototypes and $x_j$ a chosen attribute.
Ensure: Index of prototype that attracts $e$.
 1: for $i = 1$ to $lenght(ProtList)$ do
 2:     $v \leftarrow attr_{x_j}(e)$
 3:     if $\forall k \geq i, distance(attr_{x_j}(ProtList[i], v) \leq distance(attr_{x_j}(ProtList[k]), v)$
        then
 4:         return $i$
 5:     end if
 6: end for
```

Algorithm 3 $Compute\_Prototypes(x_j, S, m, ms, C)$

Require: x_j is the attribute, S is the dataset, m is the maximum children of prototypes,ms is the metric space, C is the set of classes.

Ensure: Multidimensional ranked list of prototypes.

```
1: for all class $c \in C$ do
2:     $S_c \leftarrow \{e \in S : class(e) = c\}$
3:     if $S_c \neq 0$ then
4:         $V_c \leftarrow Values(x_j, S_c)$
5:         for all element $v \in V_c$ do
6:             $MeanDistance_c[v] \leftarrow \frac{\sum_{i \in V_c} distance(v,i) * Card(i, S_c, x_j)}{|S_c|}$
7:         end for
8:     end if
9: end for
10: $UV \leftarrow 0$
11: $ProtList \leftarrow 0$
12: $RC \leftarrow C$
13: $Values \leftarrow Values(x_j, S)$
14: $Prots \leftarrow min(|C|, m, Values)$
15: for $k = 1$ to $Prots$ do
16:     $BestProt \leftarrow Argmin_e\{MeanDistance_c[Attr_{x_j}(e)]\}_{c \in RC, e \in S, Attr_{x_j}(e) \notin UV}$
17:     $RC \leftarrow RC - Class(BestProt)$
18:     $ProtList \leftarrow append(ProtList, BestProt)$
19:     $UV \leftarrow UV \cup Attr_{x_j}(\text{BestProt})$
20: end for
21: for $i = 1$ to $length(ProtList)$ do
22:     $\hat{S}_i \leftarrow \{e \in S : i = Attracts(e, ProtList, x_j)\}$
23:     $ProtList_i \leftarrow ProtList_i \cup \hat{S}_i$
24: end for
25: return ProtList
```

those of nominals. For instance, in order to select the best partition for a numerical attribute, C4.5 evaluates all the intermediate cut points obtained from the values of this attribute in the dataset. Analogously, in the case of nominal attributes, C4.5 evaluates one split for each possible attribute value. So, this makes our splitting criteria more efficient than those used in classical decision tree learning algorithms. Additionally, it is important to note that distances are computed between attribute values and not between examples. This issue is also crucial for efficiency.

The previous functions are used for the generation of our learning system. The last two functions which will be described are those that provide new functionality to the classifying process for the instances. The first one, *ProbTree* (Algorithm 4), is responsible for calculating, for each examples used to test the system, the probabilities associated with each tree node. As explained throughout this article, we use Newton Tree to estimate probabilities in a stochastic way. With $\hat{p}(\nu, e)$ we denote the probability that e falls into node ν (coming from its parent), which is derived from the *attraction* function.

Finally, *ProbClass* (see Algorithm 5) returns a stochastic probability vector estimation which is obtained crossing the Newton Tree (with probabilities calculated with the above function), from the leaves to root.

Algorithm 4 $ProbTree(e, Children(\nu))$

```
Require: $e$ an example, $Children(\nu)$ a set of succesors nodes (Prototypes).
1: if $|Children(v)| = 0$ then
2:     return
3: else
4:     for all $k \in Children(\nu)$ do
5:         $\hat{p}(k, e) \leftarrow \frac{attraction(e,k)}{\sum_{\mu \in Children(Parent(k))} attraction(e,\mu)}$
6:         $ProbTree(e, (Children(k))$
7:     end for
8: end if
```

Algorithm 5 $ProbClass(e, Children(\nu))$

```
Require: $Children(\nu)$ a set of succesors nodes (Prototypes).
Ensure: a set of class probabilities in the root of the tree.
1: if $|Children(\nu)| = 0$ then
2:     $\overrightarrow{p}(\nu, e) \leftarrow \langle Laplace(1, \nu), ..., Laplace(c, \nu)\rangle \cdot \hat{p}(\nu, e)$
3:     return $\overrightarrow{p}(\nu, e)$
4: else
5:     for all $k \in Children(\nu)$ do
6:         $\overrightarrow{p}(\nu, e) \leftarrow \overrightarrow{p}(v, e) + ProbClass(e, Children(k))$
7:     end for
8: end if
9: return $\overrightarrow{p}(\nu, e) \cdot \hat{p}(\nu, e)$
```