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 the graphical representation of 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). We present a way of graphically representing the original stochastic probability estimation trees using a user-friendly gravitation simile.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 [20] 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) [18][6], 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 [18] or through a pruning-smoothing [6]. 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.

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 [14][9][17]. We will present the details of Newton trees 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* attributes but just provide a distance for each datatype.
- We use medoids (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 reduce the number of partitions to evaluate (see Section 3.2).
- We provide a graphical representation of the trees to easily interpret them.
- We evaluate the trees using a qualitative measure of error (accuracy), a measure of ranking quality (AUC, *Area Under the ROC Curve*) and a measure of calibration and refinement quality (MSE, *Mean Squared Error*).

The paper is organised as follows. Section 2 introduces notation and basic terminology on decision tree learning and probability estimation trees, and also reviews some related work. Section 3 introduces Newton Trees, by first describing the attraction function and then explaining how trees are learned and used to obtain the probability estimations. It also introduces a user-friendly representation of Newton trees. Section 4 includes a set of experiments, which compare Newton Trees with a common PET (C4.5 without pruning and Laplace estimation). Finally, Section 5 presents the conclusions and the future work.

2 Notation and Previous Work

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 $g : E \rightarrow \{true, false\}$. 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. A PET is a decision tree which outputs a probability for each class.

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 (although there are exceptions [4, 6, 15]). 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 tree are frequently (but not always) called stochastic decision trees (e.g. [12]).

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 [15], 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, by using Kernel Density Estimates. In other words, a tree can be constructed by a classical algorithm (such as C4.5 [19] or CART [3]) and its probabilistic or stochastic interpretation can be inconsistent to the way the decision tree was constructed.

A third issue is how different datatypes are handled. Many of the previous approaches only deal with numerical attributes ([1], [2]) or only 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 [19] or CART [3]. Even in the case of fuzzy decision trees (which often 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 (structured) datatypes such as intervals, sequences or sets.

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.

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 [22] 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 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 medoid (the element in each cluster such that 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 conditional probabilities by mere normalisation, as shown in Figure 1 (right).

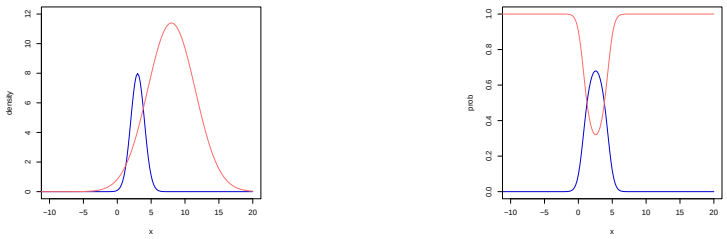

Fig. 1. (Left) Two normal distributions placed at centres 3 and 8, with standard deviations 1 and 3.5 (respectively) and masses 20 and 100 (respectively). (Right) The conditional probabilities derived from the Gaussians.

The problem of the previous approach is that when masses are too disparate, one distribution can cover the other, giving a plain partition where all the elements go to one prototype. One criterion to avoid this is to give extra importance

to distance, so that at distance 0 the probability is always 1. A way to do this is to employ an inverse-square law such as in gravitation. Hence, 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 $d(e, \pi) = d$:

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

We are interested in deriving class probabilities by 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).

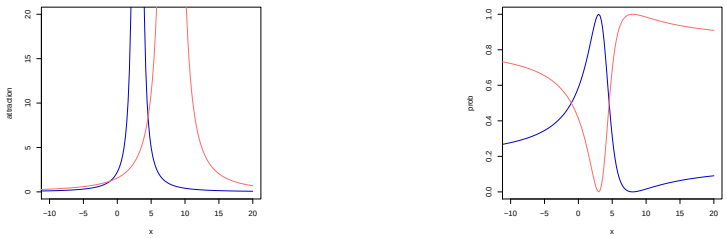

Fig. 2. (Left) Two gravitational centres at 3 and 8 with masses 20 and 100 (respectively). (Right) The probabilities derived from the gravitational centres.

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 is easy to see that the attraction to 8 will be higher than the attraction to 3 for a point placed at -100.

Of course, the idea of using the gravitational law in machine learning is not new at all, for instance in clustering [9] or classification [17]. The same Inverse Square Law principle is presented 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

Centre splitting [21] is a machine learning method which consists in dividing the input space in different regions where each region is represented by a centre[1]. In every iteration, 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 examples are managed as a whole.This appreciation leads us to propose a decision tree inference strategy where centroids are computed by considering only the values of one attribute, which allows us to join centre splitting and decision tree learning techniques in an elegant way.

[1] The centre may match to an existing example or not

The detailed definition of the algorithm can be found in [16]. Here, we give a more sketchy description: 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 [19]). Then, the split proceeds by associating every instance to its closest attribute prototype, which typically produces impure clusters[2]. Although the computation of distances is quadratic on the number of instances, we can reduce it by using a distance matrix per attribute (of size $n_r \times n_r$, where n_r is the number of different attribute values) prior to the algorithm execution. 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.

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 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 correction as has been shown in [18, 6]. 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

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

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

[2] 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.

The stochastic calculation of the probabilities seen above may seem too cryptic for a general use of these trees if intelligibility is a requirement. 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 3 (left) shows this user-friendly representation of a Newton Tree for the Hepatitis dataset from the UCI repository [8]. 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 are represented as balls of a size which is proportional to the node mass (for instance, the node with a mass of 17 represents that 17 training examples fall into it). The ball also shows the proportion of examples of each class in different colours. Additionally, the value for the attribute prototype is shown in the middle of each ball. 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 3 (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 in the Newton Trees representation may help users understand the way in that probabilities are estimated, making Newton trees less cryptic than other PET methods.

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 [18][6]. In particular, we chose J48 (the variant of C45.) implemented in Weka [10]. We used *Gain ratio* as splitting criterion for Newton trees and J48. The evaluation was performed over 30 datasets from the UCI repository [8], from which we removed instances with missing values (see [16] 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 ([7]): accuracy, as a qualitative measure of error, AUC (*Area Under the Curve*) as a measure of ranking quality, (using Hand & Till's multiclass version [11]) and MSE (*Mean Squared Error*) as a measure of calibration and refinement quality.

Table 1 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 significant, we used the Wilcoxon signed-ranks test with a confidence level of $\alpha = 0.05$ and $N = 30$ data sets, as suggested in [5]. Significant differences are shown in bold. Finally, in Table 2 we focus on these differences, showing an entry $w/t/l$ for each measure and dataset subset, which indicates that Newton

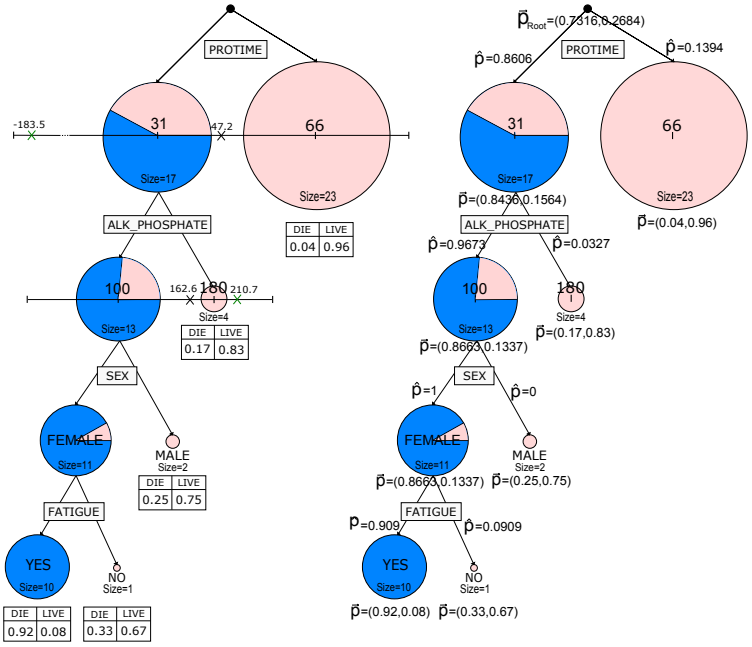

Fig. 3. (Left) Newton Tree for the hepatitis dataset. (Right) The node probability vectors, children probabilities and global probability vector for example (PROTIME=40, ALK_PH=120, SEX=FEMALE, FATIGUE= UNKNOWN)

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 1, 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 ([13]). If we look at the significance results in Table 2, 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 Conclusions and Future Work

This paper has presented a novel probability estimation tree learning method which is based on computing prototypes 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

Name	Classes	Att Type	Newton Trees			Unpruned Laplace J48		
			Acc.	AUC	MSE	Acc.	AUC	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 1. 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 2. Aggregated results using the statistical tests

understood. 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, 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. How-

ever, these two modifications would entail extra computational cost and could only be justified if there is a significant improvement in the results.

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. Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Chapman & Hall, New York, NY, 1984.
4. W. Buntine. Learning classification trees. *Stats. and Computing*, 2(2):63–73, 1992.
5. J. Demsar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, January 2006.
6. 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.
7. C. Ferri, J. Hernández-Orallo, and R. Modroiu. An experimental comparison of performance measures for classification. *Pattern Recogn. Lett.*, 30(1):27–38, 2009.
8. A. Frank and A. Asuncion. UCI machine learning repository, 2010.
9. 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.
10. M. Hall, Frank E.and G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software. *SIGKDD Explorations*, 11(1):10–18, 2009.
11. D.J. Hand and R.J. Till. A simple generalisation of the area under the ROC curve for multiple class classification problems. *Machine Learning*, 45(2):171–186, 2001.
12. R.F. Hespos and P.A. Strassmann. Stochastic decision trees for the analysis of investment decisions. *Management Science*, 11(10):244–259, 1965.
13. Jin Huang and Charles X. Ling. Using auc and accuracy in evaluating learning algorithms - appendices. *IEEE Trans. Knowl. Data Eng.*, 17(3), 2005.
14. M. Indulska and ME Orlowska. Gravity based spatial clustering. In *Proc. Int. Sym. on Advances in geographic information systems*, page 130, 2002.
15. C.X. Ling and R.J. Yan. Decision tree with better ranking. In *International Conference on Machine Learning*, volume 20-2, page 480, 2003.
16. 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.
17. L. Peng, B. Yang, Y. Chen, and A. Abraham. Data gravitation based classification. *Information Sciences*, 179(6):809–819, 2009.
18. Foster J. Provost and Pedro Domingos. Tree induction for probability-based ranking. *Machine Learning*, 52(3):199–215, 2003.
19. J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.
20. Lior Rokach and Oded Maimon. *Data Mining with Decision Trees: Theory and Applications*. World Scientific, 2008.
21. C.J. Thornton. *Truth from trash: how learning makes sense*. The MIT Press, 2000.
22. Berwin A. Turlach. Bandwidth selection in kernel density estimation: A review. In *CORE and Institut de Statistique*, 1993.