

Decision Trees for Uncertain Data

Smith Tsang[†], Ben Kao[†], Kevin Y. Yip[‡], Wai-Shing Ho[†], Sau Dan Lee[†]

[†]*Department of Computer Science, The University of Hong Kong, Hong Kong*
pktsang, kao, wsho, sdlee@cs.hku.hk

[‡]*Department of Computer Science, Yale University, U.S.A.*
yuklap.yip@yale.edu

Abstract—Traditional decision tree classifiers work with data whose values are known and precise. We extend such classifiers to handle data with uncertain information, which originates from measurement/quantisation errors, data staleness, multiple repeated measurements, etc. The value uncertainty is represented by multiple values forming a probability distribution function (pdf). We discover that the accuracy of a decision tree classifier can be much improved if the whole pdf, rather than a simple statistic, is taken into account. We extend classical decision tree building algorithms to handle data tuples with uncertain values. Since processing pdf's is computationally more costly, we propose a series of pruning techniques that can greatly improve the efficiency of the construction of decision trees.

I. INTRODUCTION

Classification is a classical problem in machine learning and data mining[1]. Given a set of training data tuples, each having a class label and being represented by a feature vector, the task is to algorithmically build a model that predicts the class label of an unseen test tuple based on the tuple's feature vector. One of the most popular classification models is the decision tree model. Decision trees are popular because they are practical and easy to understand. Rules can also be extracted from decision trees easily. Many algorithms, such as ID3[2] and C4.5[3] have been devised for decision tree construction. These algorithms are widely adopted and used in a wide range of applications such as image recognition, medical diagnosis[4], credit rating of loan applicants, scientific tests, fraud detection, and target marketing.

In traditional decision-tree classification, a feature (an attribute) of a tuple is either categorical or numerical. For the latter, a precise and definite *point value* is usually assumed. In many applications, however, data uncertainty is common. The value of a feature/attribute is thus best captured by not a single point value, but by a range of values giving rise to a probability distribution. Data uncertainty arises naturally in many applications due to various reasons: measurement errors, data staleness, repeated measurements, limitations of the data collection process, etc.

A simple way to handle data uncertainty is to abstract probability distributions by summary statistics such as means and variances. We call this approach *Averaging*. Another approach is to consider the complete information carried by the probability distributions to build a decision tree. We call this approach *Distribution-based*.

In this paper we study the problem of constructing decision tree classifiers on data with uncertain numerical attributes. Our

goals are (1) to devise an algorithm for building decision trees from uncertain data using the Distribution-based approach; (2) to investigate whether the Distribution-based approach could lead to a higher classification accuracy compared with the Averaging approach; and (3) to establish a theoretical foundation on which pruning techniques are derived that can significantly improve the computational efficiency of the Distribution-based algorithms.

II. RELATED WORKS

There has been a growing interest in uncertain data mining. The well-known k-means clustering algorithm has been extended to the UK-means algorithm[5] and various pruning techniques have been proposed[6], [7].

Decision tree classification with missing data has been addressed for decades in the form of missing values[2], [3]. In C4.5[3], these are handled by using *fractional tuples*. In this work, we adopt the technique of fractional tuple for splitting tuples into subsets when the domain of its pdf spans across the split point. However, handling data values represented as pdf's is unprecedented.

In fuzzy decision tree classification, both attributes and class labels can be fuzzy and are represented in fuzzy terms[8]. Given a fuzzy attribute of a data tuple, a degree (called membership) is assigned to each possible value, showing the extent to which the data tuple belongs to a particular value. Our work instead gives classification results as a distribution: for each test tuple, we give a distribution telling how likely it belongs to each class.

Building a decision tree on tuples with numerical, point-valued data is computationally demanding[9]. Finding the best split point is computationally expensive. To improve efficiency, many techniques have been proposed to reduce the number of candidate split points[10], [9], [11]. Our work can be considered an extension of these optimisation techniques.

III. PROBLEM DEFINITION

In our model, a dataset consists of d *training* tuples, $\{t_1, t_2, \dots, t_d\}$, and k numerical (real-valued) feature attributes, A_1, \dots, A_k . Each tuple t_i is associated with a feature vector $V_i = (f_{i,1}, f_{i,2}, \dots, f_{i,k})$ and a class label $c_i \in C$. Here, each $f_{i,j}$ is a pdf with domain $[a_{i,j}, b_{i,j}]$ modelling the uncertain value of attribute A_j in tuple t_i . The classification problem is to construct a model M that maps each feature vector $(f_{x,1}, \dots, f_{x,k})$ to a probability distribution P_x on C

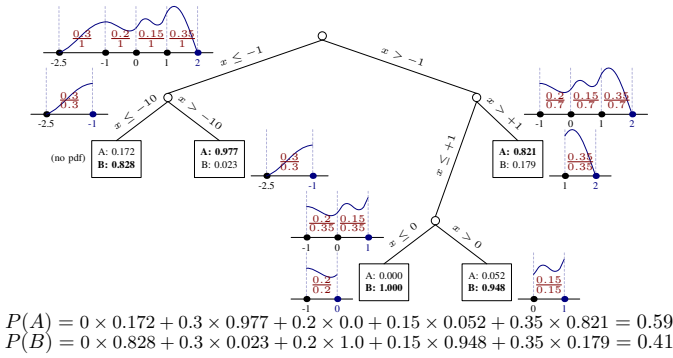


Fig. 1. Classifying a test tuple

such that given a *test* tuple $t_0 = (f_{0,1}, \dots, f_{0,k}, c_0)$, $P_0 = M(f_{0,1}, \dots, f_{0,k})$ predicts the class label c_0 with high accuracy. We say that P_0 predicts c_0 if $c_0 = \arg \max_{c \in C} \{P_0(c)\}$.

We consider *binary* decision trees with tests on numerical attributes. Each internal node n of a decision tree is associated with an attribute A_{j_n} and a split point z_n , giving a binary test $x \leq z_n$. An internal node has exactly 2 children. Each leaf node m in the decision tree is associated with a discrete probability distribution P_m over C .

To determine the class label of a given *test* tuple $t_0 = (f_{0,1}, \dots, f_{0,k}, ?)$, we traverse the tree top down, starting from the root node. When we visit an internal node n , we split the tuple into two parts at z_n and distribute each part recursively down the child nodes accordingly. Eventually, we reach leaf nodes. The probability distribution P_m at each leaf node m contributes¹ to the final distribution P_0 for predicting the class label of t_0 . This is illustrated with the example in Figure 1.

A pdf $f_{i,j}$ could be programmed analytically if it can be specified in closed forms. More typically, it would be implemented numerically by storing a set of s sample points $x \in [a_{i,j}, b_{i,j}]$ with the associated value $f_{i,j}(x)$, effectively approximating $f_{i,j}$ by a discrete distribution. We adopt this numerical approach for the rest this paper. With this representation, the amount of information available is exploded by a factor of s . Hopefully, the richer information allows us to build a better classification model.

The most challenging task is to construct a decision tree based on tuples with uncertain values, finding suitable A_{j_n} and z_n for each internal node n , as well as an appropriate probability distribution P_m for each leaf node m .

IV. ALGORITHMS

We propose two approaches for handling uncertain data.

A. Averaging

A straight-forward way to deal with the uncertain information is to replace each pdf with its expected value, thus effectively converting the data tuples to point-valued tuples. This reduces the problem back to that for point-valued data, and hence traditional decision tree algorithms such as ID3

¹ Details are omitted due to the lack of space.

TABLE I
EXAMPLE TUPLES

tuple	class	mean	probability distribution				
			-10	-1.0	0.0	+1.0	+10
1	A	+2.0		8/11			3/11
2	A	-2.0	1/9	8/9			
3	A	+2.0		5/8		1/8	2/8
4	B	-2.0	5/19	1/19		13/19	
5	B	+2.0			1/35	30/35	4/35
6	B	-2.0	3/11			8/11	

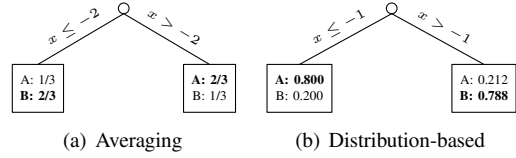


Fig. 2. Decision tree built from example tuples in Table I

and C4.5[3] can be reused. We call this approach **AVG** (for Averaging). We use an algorithm based on C4.5, using entropy as the dispersion measure. To alleviate the problem of overfitting, we apply the techniques of *pre-pruning* and *post-pruning* (see [12], [3]). This is illustrated using the example tuples shown in Table I. The resulting decision tree is shown in Figure 2(a). Now, if we use the 6 tuples in Table I as test tuples, this decision tree will classify tuples 2, 4, 6 as class “B” (the most likely class label in L) and the rest as “A”. Hence it misclassifies tuples 2 and 5. The accuracy is $2/3$.

B. Distribution-based

For uncertain data, we adopt the same decision tree building framework, including the techniques of *pre-pruning* and *post-pruning*. After an attribute A_{j_n} and a split point z_n has been chosen for a node n , we split the set of tuples S into two subsets L and R . The major difference from the point-data case lies in the way the set S is split. If the pdf properly contains the split point, i.e., $a_{i,j_n} \leq z_n < b_{i,j_n}$, we split t_i into two *fractional tuples*[3] t_L and t_R and add them to L and R , respectively.¹ We call this algorithm **UDT** (for Uncertain Decision Tree).

The key to building a good decision tree is a good choice of an attribute A_{j_n} and a split point z_n for each node n . With uncertain data, however, the number of choices of a split point given an attribute is not limited to $m - 1$ point values, but the union of the domains of all pdfs $f_{i,j_n} \forall i = 1, \dots, m$. Representing each $f_{i,j}$ with s sample points, there are in total ms sample points. So, there are at most $ms - 1$ possible split points to consider. Comparing to **AVG**, **UDT** is s time more expensive, computationally.

Let us re-examine the example tuples in Table I to see how the distribution-based algorithm can improve classification accuracy. By taking into account the probability distribution, **UDT** builds the tree shown in Figure 3 before *pre-pruning* and *post-pruning* are applied. This tree turns out to have a 100% classification accuracy! After *post-pruning*, we get the tree in Figure 2(b). Use the 6 tuples in Table I as testing tuples to test this pruned tree, all 6 tuples are classified correctly.

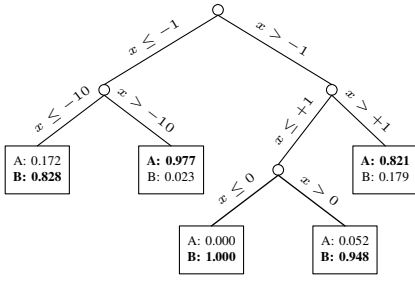


Fig. 3. Example decision tree before post-pruning

TABLE II
ACCURACY IMPROVEMENT BY CONSIDERING THE DISTRIBUTION

Data Set	AVG	UDT				
		Best	$w = 1\%$	$w = 5\%$	$w = 10\%$	$w = 20\%$
JapaneseVowel	81.89	87.30	* 87.30 (distribution based raw data)			
Pen-Digit	90.87	95.22	91.66	92.18	93.79	* 95.22
PageBlock	95.73	96.82	* 96.82	96.32	95.74	94.87#
Satellite	84.48	87.73	85.18	87.10	* 87.73	86.25
Segment	89.37	92.91	91.91	* 92.91	92.23	89.11#
Vehicle	71.03	75.09	72.44	72.98	73.18	* 75.09
BreastCancer	93.52	95.93	94.73	94.28	95.51	* 95.93
Ionosphere	88.69	91.69	89.65	88.92	* 91.69	91.60
Glass	66.49	72.75	69.60	* 72.75	70.79	69.69
Iris	94.73	96.13	94.47#	95.27	96.00	* 96.13

The accuracy is 100%. This example thus illustrates that by considering probability distributions rather than just expected values, we can potentially build a more accurate decision tree.

C. Experiments on Accuracy

We have implemented AVG and UDT and applied them to 10 real data sets taken from the UCI Machine Learning Repository[13]. The results are shown in Table II. For most of the datasets, the data uncertainty is modelled with a Gaussian distribution with a controllable parameter w . For the “JapaneseVowel” data set, we use the uncertainty given by the raw data to model the pdf.

From the table, we see that UDT builds more accurate decision trees than AVG does. For instance, for the first data set, whose pdf is modelled from the raw data samples, the accuracy is improved from 81.89% to 87.30%; i.e., the error rate is reduced from 18.11% down to 12.70%, which is a very significant improvement. Only in a few cases (marked with “#” in the table) does UDT give slightly worse accuracies than AVG. Comparing the second and third columns of Table II, we see that UDT can potentially build remarkably more accurate decision trees than AVG.

V. PRUNING ALGORITHMS

Although UDT can build a more accurate decision tree, it is not as efficient as AVG. UDT has to perform s times as many computations as AVG. We have come up with a few strategies for pruning candidate split points.

A. Pruning Empty and Homogeneous Intervals

The hardest problem to solve in UDT is to select an attribute A_j and split point z_j to minimise the entropy. Let us focus on finding the best split point for one particular attribute A_j .

We define the set of *end-points* of tuples in S on attribute A_j as $Q_j = \{q \mid (q = a_{h,j}) \vee (q = b_{h,j}) \text{ for some } t_h \in S\}$. We assume that there are v such end-points, q_1, q_2, \dots, q_v , sorted in ascending order. Within $[q_1, q_v]$, we want to find an optimal split point for attribute A_j .

Definition 1: For a given set of tuples S , an *optimal split point* for an attribute A_j is one that minimises the entropy. (Note that the minimisation is taken over all $z \in [q_1, q_v]$.)

The end-points define $v - 1$ disjoint intervals: $(q_i, q_{i+1}]$ for $i = 1, \dots, v - 1$. We will examine each interval separately. For convenience, an interval is denoted by $(a, b]$.

Definition 2 (Empty interval): An interval $(a, b]$ is empty if $\int_a^b f_{h,j}(x) dx = 0$ for all $t_h \in S$.

Definition 3 (Homogeneous interval): An interval $(a, b]$ is homogeneous if there exists a class label $c \in C$ such that $\int_a^b f_{h,j}(x) dx \neq 0 \implies c_h = c$ for all $t_h \in S$.

Intuitively, an interval is empty if no pdf’s domain intersects it; an interval is homogeneous if all the pdf’s that intersect it come from tuples of the same class.

Definition 4 (Heterogeneous interval): An interval $(a, b]$ is heterogeneous if it is neither empty nor homogeneous.

Theorem 1: If an optimal split point falls in an empty interval, then an end-point of the interval is also an optimal split point.

Theorem 2: If an optimal split point falls in a homogeneous interval, then an end-point of the interval is also an optimal split point.

The implication of these theorems is that interior points in empty and homogeneous intervals need not be considered when we are looking for an optimal split point. The analogue for the point-data case is also well known. [10].

Applying Theorems 1 and 2 to UDT allows us to prune away the interior points of empty and homogeneous intervals. This gives our *Basic Pruning* algorithm UDT-BP.

B. Pruning by Bounding

Our next algorithm attempts to prune away heterogeneous intervals through a bounding technique. First we compute the entropy $H(q, A_j)$ for all end-points $q \in Q_j$. Let H_j^* be the minimum value. Next, for each heterogeneous interval $(a, b]$, we compute a lower bound, L_j , of $H(z, A_j)$ over all candidate split points $z \in (a, b]$. If $L_j \geq H_j^*$, we know that none of the candidate split points within the interval $(a, b]$ can give an entropy that is smaller than H_j^* and thus the whole interval can be pruned.

We note that the number of end-points is much smaller than the total number of candidate split points. So, if a lot of heterogeneous intervals are pruned in this manner, we can eliminate many entropy calculations. The cost¹ of computing L_j is roughly the same as evaluating the entropy of *only one* split point. So, if an interval is pruned by the lower-bound technique, we have reduced the cost of computing the entropy values of all split points in the interval to the computation of one entropy-like lower bound. Combining this heterogeneous interval pruning technique with those for empty and homogeneous intervals gives us the *Local Pruning*

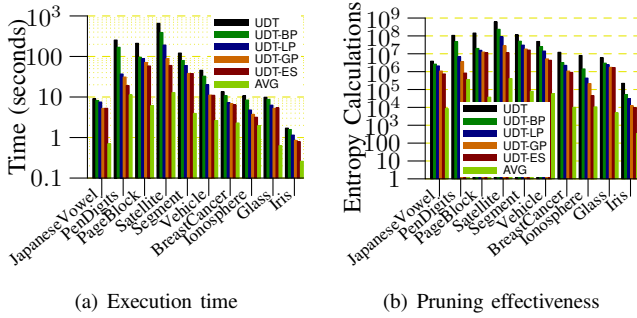


Fig. 4. Performance of the pruning algorithms

algorithm UDT-LP. A simple but effective improvement on UDT-LP is to use a global (across all attributes A_j) threshold $H^* = \min_{1 \leq j \leq k} H_j^*$ for pruning. This gives UDT-GP.

C. End-point sampling

As we will see later in Section VI, UDT-GP is very effective in pruning intervals. In some settings, UDT-GP reduces the number of “entropy calculations” (including the calculation of entropy values of split points and the calculation of entropy-like lower bounds for intervals) to only 2.7% of that of UDT. On a closer inspection, we find that many of these remaining entropy calculations come from the determination of end-point entropy values. In order to further improve the algorithm’s performance, we propose a method to prune these end-points.

We can take a sample of the end-points (say 10%) and use their entropy values to derive a pruning threshold. This threshold might be slightly less effective as the one derived from all end-points, however, finding it requires much fewer entropy calculations. Incorporate this *End-point Sampling* strategy into UDT-GP gives us our next algorithm UDT-ES.

VI. EXPERIMENTS ON EFFICIENCY

The algorithms described above have been implemented in Java using JDK 1.6 and a series of experiments were performed on a PC with an Intel Core 2 Duo 2.66GHz CPU. The data sets used are the same as those used in Section IV-B.

A. Execution Time

We first examine the execution time of the algorithms, which is charted in Figure 4(a). We have given also the execution time of the AVG algorithm (see Section IV-A). Note that AVG builds different decision trees from those constructed by the UDT-based algorithms, and that AVG generally builds less accurate classifiers. The execution time of AVG shown in the figure is for reference only. From the figure, we observe the following general (ascending) order of efficiency: UDT, UDT-BP, UDT-LP, UDT-GP, UDT-ES. The AVG algorithm, which does not exploit the uncertainty information, takes the least time to finish, but cannot achieve as high an accuracy compared to the distribution-based algorithms (see Section IV-C). We remark that in the experiment, each pdf is represented by 100 sample points (i.e., $s = 100$). All UDT-based algorithms thus have to handle 99 times more data (except for the

“JapaneseVowel” data) than AVG, which only processes one average per pdf.

B. Pruning Effectiveness

Next, we study the pruning effectiveness of the algorithms. Figure 4(b) shows the number of entropy calculations performed by each algorithm. As we have explained, the computation time of the lower bound of an interval is comparable to that of computing an entropy. Therefore, for UDT-LP, UDT-GP, and UDT-ES, the number of entropy calculations include the number of lower bounds computed. The figure shows that our pruning techniques introduced in Section V are highly effective. Indeed, UDT-ES reduces the number of entropy calculations to 0.56%–28% when compared with UDT. It thus achieves a pruning effectiveness ranging from 72% up to as much as 99.44%. As entropy calculations dominate the execution time of UDT, such effective pruning techniques significantly reduce the tree-construction time.

VII. CONCLUDING REMARKS

We have extended the model of decision-tree classification and tree-construction algorithms[3] to accommodate data tuples having numerical attributes with uncertainty described by arbitrary pdf’s. Experiments show that exploiting data uncertainty leads to decision trees with remarkably higher accuracies. Performance is an issue, though, because of the increased amount of information to be processed. We have devised a series of highly effective pruning techniques to improve tree construction efficiency. Pruning by bounding and end-point sampling are novel pruning techniques.

Although our novel techniques are primarily designed to handle uncertain data, they are also useful for building decision trees using classical algorithms when there are tremendous amounts of data tuples.

Acknowledgement: This research is supported by Hong Kong Research Grants Council Grant HKU 7134/06E.

REFERENCES

- [1] R. Agrawal, T. Imielinski, and A. N. Swami, “Database mining: A performance perspective,” *IEEE Trans. Knowl. Data Eng.*, 1993.
- [2] J. R. Quinlan, “Induction of decision trees,” *Machine Learning*, 1986.
- [3] —, *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
- [4] C. L. Tsien, I. S. Kohane, and N. McIntosh, “Multiple signal integration by decision tree induction to detect artifacts in the neonatal intensive care unit,” *Artificial Intelligence in Medicine*, vol. 19, no. 3, 2000.
- [5] M. Chau, R. Cheng, B. Kao, and J. Ng, “Uncertain data mining: An example in clustering location data,” in *PAKDD*, 2006, pp. 199–204.
- [6] W. K. Ngai, B. Kao, C. K. Chui, R. Cheng, M. Chau, and K. Y. Yip, “Efficient clustering of uncertain data,” in *ICDM*, 2006, pp. 436–445.
- [7] S. D. Lee, B. Kao, and R. Cheng, “Reducing UK-means to K-means,” in *1st Workshop on Data Mining of Uncertain Data*, in *ICDM*, 2007.
- [8] Y. Yuan and M. J. Shaw, “Induction of fuzzy decision trees,” *Fuzzy Sets Syst.*, vol. 69, no. 2, pp. 125–139, 1995.
- [9] T. Elomaa and J. Rousu, “General and efficient multisplitting of numerical attributes,” *Machine Learning*, vol. 36, no. 3, pp. 201–244, 1999.
- [10] U. M. Fayyad and K. B. Irani, “On the handling of continuous-valued attributes in decision tree generation,” *Machine Learning*, 1992.
- [11] T. Elomaa and J. Rousu, “Efficient multisplitting revisited: Optima-preserving elimination of partition candidates,” *Data Mining and Knowledge Discovery*, vol. 8, no. 2, pp. 97–126, 2004.
- [12] T. M. Mitchell, *Machine Learning*. McGraw-Hill, 1997.
- [13] A. Asuncion and D. Newman, “UCI machine learning repository,” 2007. [Online]. Available: <http://www.ics.uci.edu/~mllearn/MLRepository.html>