Resolution Trees for Tentative Information

[1]V. Redya Jadav, [2]M. Nageswara Rao, [3]Ella Sarada
[1,2,3]Department of CSE, Bomma Institute of Technology and Science, Khammam

Abstract
Traditional decision tree classifiers work with data whose values are known and precise. We extend classical decision tree building algorithms to handle data tuples with uncertain values. Extensive experiments have been conducted which show that the resulting classifiers are more accurate than those using value averages. Since processing pdfs is computationally more costly than processing single values (e.g.,averages), decision tree construction on uncertain data is more CPU demanding than that for certain data.We extend such classifiers to handle data with uncertain information. Value uncertainty arises in many applications during the data collection process. Example sources of uncertainty include measurement/quantization errors, data staleness, and multiple repeated measurements.With uncertainty, the value of a data item is often represented not by one single value, but by multiple values forming a probability distribution. Rather than abstracting uncertain data by statistical derivatives (such as mean and median), we discover that the accuracy of a decision tree classifier can be much improved if the "complete information" of a data item (taking into account the probability density function (pdf)) is utilized. To tackle this problem, we propose a series of pruning techniques that can greatly improve construction efficiency.

Keywords
???????????? IS MISSING ??????????

I. Introduction

CLASSIFICATION is a classical problem in machine learning and data mining. 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 and C4.5, 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 not by a single point value, but by a range of values giving rise to a probability distribution. 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.

A. Measurement Errors

Data obtained from measurements by physical devices are often imprecise due to measurement errors. As an example, a tympanic (ear) thermometer measures body temperature by measuring the temperature of the eardrum via an infrared sensor. A typical ear thermometer has a quoted calibration error of 0:2 C, which is about 6.7 percent of the normal range of operation, noting that the human body temperature ranges from 37 C (normal) and to 40 C (severe fever).Compound that with other factors such as placement and technique, measurement error can be very high. For example, it is reported in [5] that about 24 percent of measurements are off by more than 0:5 C, or about 17 percent of the operational range.

B. Data Staleness

In some applications, data values are continuously changing and recorded information is always stale. One example is location-based tracking system. The where about of a mobile device can only be approximated by imposing an uncertainty model on its last reported location. A typical uncertainty model requires knowledge about the moving speed of the device and whether its movement is restricted (such as a car moving on a road network) or unrestricted (such as an animal moving on plains). Typically, a 2D probability density function is defined over a bounded region to model such uncertainty.

C. Repeated Measurements

Perhaps the most common source of uncertainty comes from repeated measurements. For example, a patient's body temperature could be taken multiple times during a day; an anemometer could record wind speed once every minute; the space shuttle has a large number of heat sensors installed all over its surface. As a more elaborate example, consider the "BreastCancer" data set reported in. This data set contains a number of tuples. Each tuple corresponds to a microscopic image of stained cell nuclei. A typical image contains 10-40 nuclei.One of the features extracted from each image is the average radius of nuclei. We remark that such a radius measure contains a few sources of uncertainty: 1) an average is taken from a large number of nuclei from an image, 2) the radius of an (irregularly shaped) nucleus is obtained by averaging the length of the radial line segments defined by the centroid of the nucleus and a large number of sample points on the nucleus' perimeter, and 3) a nucleus' perimeter was outlined by a user over a fuzzy 2D image.

Yet another source of uncertainty comes from the limitation of the data collection process. For example, a survey may ask a question like, "How many hours of TV do you watch each week?" A typical respondent would not reply with an exact precise answer. Rather, a range (e.g., "6-8 hours") is usually replied, possibly because the respon-dent is not so sure about the answer himself. In this example, the survey can restrict an answer to fall

into a few preset categories (such as "2-4 hours," "4-7 hours," etc.). However, this restriction unnecessarily limits the respondents' choices and adds noise to the data. Also, for preserving privacy, sometimes, point data values are transformed to ranges on purpose before publication.

From the above examples, we see that in many applications, information cannot be ideally represented by point data. More often, a value is best captured by a range possibly with a pdf. Our concept of uncertainty refers to such ranges of values. Again, our goal is to investigate how decision trees are built over uncertain (range) data. Our contributions include the following:

1. A basic algorithm for constructing decision trees out of uncertain data sets,
2. A study comparing the classification accuracy achieved by the Averaging approach and the Distribution-based approach,
3. A set of mathematical theorems that allows significant pruning of the large search space of the best split point determination during tree construction,
4. Efficient algorithms that employ pruning techniques derived from the theorems, and
5. A performance analysis on the various algorithms through a set of experiments.

In the rest of the paper, we first describe some related works briefly in Section II. Then, we define the problem formally in Section III. In Section IV, we present our proposed algorithm and show empirically that it can build decision trees with higher accuracies than using only average values, especially when the measurement errors are modeled appropriately.

II. Related Works

There has been significant research interest in uncertain data management in recent years. Data uncertainty has been broadly classified into existential uncertainty and value uncertainty. Existential uncertainty appears when it is uncertain whether an object or a data tuple exists. For example, a data tuple in a relational database could be associated with a probability that represents the confidence of its presence. "Probabilistic databases" have been applied to semistructured data and XML. Value uncertainty, on the other hand, appears when a tuple is known to exist, but its values are not known precisely. A data item with value uncertainty is usually represented by a pdf over a finite and bounded region of possible values. One well-studied topic on value uncertainty is"imprecise queries processing". The answer to such a query is associated with a probabilistic guarantee on its correctness.

Decision tree classification on uncertain data has been addressed for decades in the form of missing values.Missing values appear when some attribute values are not available during data collection or due to data entry errors. Solutions include approximating missing values with the majority value or inferring the missing value (either by exact or probabilistic values) using a classifier on the attribute (e.g., ordered attribute tree and probabilistic attribute tree). In C4.5 and probabilistic decision trees , missing values in training data are handled by using fractional tuples. During testing, each missing value is replaced by multiple values with probabilities based on the training tuples, thus, allowing probabilistic classification results. 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.

Another related topic is fuzzy decision tree. Fuzzy information models data uncertainty arising from human perception and understanding. The uncertainty reflects the vagueness and ambiguity of concepts, e.g., how hot is "hot." In fuzzy classification, both attributes and class labels can be fuzzy and are represented in fuzzy terms. 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. There are many variations of fuzzy decision trees, e.g., fuzzy extension of ID3 and Soft Decision Tree . In these models, a node of the decision advantage of our approach is that the tuple splitting is based on probability values, giving a natural interpretation to the splitting as well as the result of classification. Building a decision tree on tuples with numerical, point-valued data is computationally demanding.

These techniques utilize the convex property of well-known evaluation functions like Information Gain and Gini Index. For the evaluation function Training Set Error (TSE), which is convex but not strictly convex, one only tree does not give a crisp test that decides deterministically In this paper, we show that accuracy can be improved by considering uncertainty information which branch down the tree a training or testing tuple is sent. Rather it gives a "soft test" or a fuzzy test on the point valued tuple. The zn 2 domðAjn Þ, giving a binary test v0;jn zn . An internal node has exactly two children, which are labeled "left" and "right", respectively. Each leaf node m in the decision tree is associated with a discrete probability distribution Pm over C. For each c 2 C, PmðcÞ gives a probability reflecting how likely a tuple assigned to leaf node m would have a class label of c.To determine the class label of a given test tuple t0 ¼ ðv0;1 ; . . . ; v0;k ; ?Þ, we traverse the tree starting from the root node until a leaf node is reached. When we visit an internal node n, we execute the test v0;j n zn and proceed to the left child or the right child accordingly. Eventually, we reach a leaf node m. The probability distribution Pm associated with m gives the probabilities that t0 belongs to each class label c 2 C. For a single result, we return the class label c 2 C that maximizes PmðcÞ.

B. Handling Uncertainty Information

Under our uncertainty model, a feature value is represented not by a single value, vi;j , but by a pdf, fi;j . For practical reasons, we assume that fi;j is nonzero only within a bounded interval ½ai;j ; bi;j . (We will briefly discuss how our methods can be extended to handle pdfs with unbounded domains in Section 7.3.) A pdf fi;j could be programmed analytically if it can be specified in closed form. More typically, it would be implemented numerically by storing a set of s sample points x 2 ½ai;j ; bi;j with the associated value fi;jðxÞ, effectively approximating fi;j by a discrete distribution with s possible values. We adopt this numerical approach for the rest of the 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. On the down side, processing large number of sample points is much more costly. In this paper, we show that accuracy can be improved by considering uncertainty information. We also propose pruning strategies that can greatly reduce the computational effort.

A decision tree under our uncertainty model resembles that of the point data model. The difference lies in the way the tree is employed to classify unseen test tuples. Similar to the training tuples, a test tuple t0 contains uncertain attributes. Its feature vector is thus a vector of pdfs ðf0;1 ; . . . ; f0;kÞ. A classification

model is thus a function M that maps such a feature vector to a probability distribution P over C. The probabilities for P are calculated as follows: During these calculations, we associate each intermediate tuple tx with a weight wx 2 ½0; 1 . Further, we recursively define the quantity nðc; tx ; wxÞ, which can be interpreted as the conditional probability that tx has class label c, when the subtree rooted at n is used as an uncertain decision tree to classify tuple tx with weight wx.

For each internal node n (including the root node), to determine nðc; tx ; wxÞ, we first check the attribute Aj n and split point zn of node n. Since the pdf of tx under attribute Aj n spans the inRerval ½ax;j n; bx;j n , we compute the "left" probability pL ¼ az x;jn fx;j n ðtÞ dt (or pL ¼ 0 in case zn < ax;j n) and the "right" probability pR ¼ 1 pL . Then we split tx into two fractional tuples tL and tR . (The concept of fractional tuples is also used in C4.5 [3] for handling missing values.) Tuples tL and tR inherit the class label of tx as well as the pdfs of tx for all attributes except Aj n . Tuple tL is assigned a weight of wL ¼ wx pL and its pdf for Aj n is given by

fL;j n

ðxÞ ¼ fx;jn ðxÞ=wL if x 2 ½ax;jn ; zn ;

0; otherwise:

Tuple tR is assigned a weight and pdf analogously. We define nðc; tx ; wxÞ ¼ pL nLðc; tL ; wLÞ þ pR nRðc; tR ; wRÞ

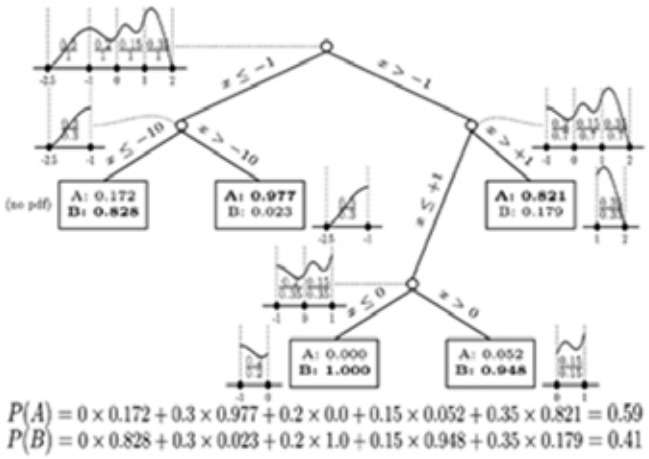


Fig. 1: Classifying a Test Tuple

where nL and nR are the left child and the right child of node n, respectively.

For every leaf node m, recall that it is associated with a probability distribution Pm over C. We define mðc; tx ; wxÞ ¼ wx PmðcÞ. Finally, for each class c, let PðcÞ ¼ rðc; t0 ; 1:0Þ, where r is the root node of the decision tree. Obtained this way, each probability PðcÞ indicates how likely it is that the test tuple t0 has class label c. These computations are illustrated in Fig. 1, which shows a test tuple t0 with one feature whose pdf has the domain ½ 2:5; 2 .

It has a weight of 1.0 and is first tested against the root node of the decision tree. Based on the split point 1, we find that pL ¼ 0:3 and pR ¼ 0:7. So, t0 is split into two tuples tL and tR with weights wL ¼ 0:3 and wR ¼ 0:7. The tuple tL inherits the pdf from t0 over the subdomain ½ 2:5; 1 , normalized by multiplying by a factor of 1=wL . Tuple tR inherits the pdf from t0 in a similar fashion. These tuples are then recursively tested down the tree until the leaf nodes are reached. The weight distributed in such a way down to each leaf node is then multiplied with the probability of each class label at that leaf node. These are finally summed up to give the probability distribution (over the class labels) for t0 , giving PðAÞ ¼ 0:59, PðBÞ ¼ 0:41.

If a single class label is desired as the result, we select the class label with the highest probability as the final answer. In fig. 1, the test tuple is thus classified as class "A" when a single result is desired.

The most challenging task is to construct a decision tree based on tuples with uncertain values. It involves finding a good testing attribute Aj n and a good split point zn for each internal node n, as well as an appropriate probability distribution Pm over C for each leaf node m. We describe algorithms for constructing such trees in the next section.

IV. Algorithms

In this section, we discuss two approaches for handling uncertain data. The first approach, called "Averaging", transforms an uncertain data set into a point-valued one by replacing each pdf with its mean value. More specifically, for each tuple ti and attribute A_j, we take distribution PLðAÞ ¼ 1=3 and PLðBÞ ¼ 2=3 over the class labels. The probability distribution of class labels in the right leaf node R is determined analogously.

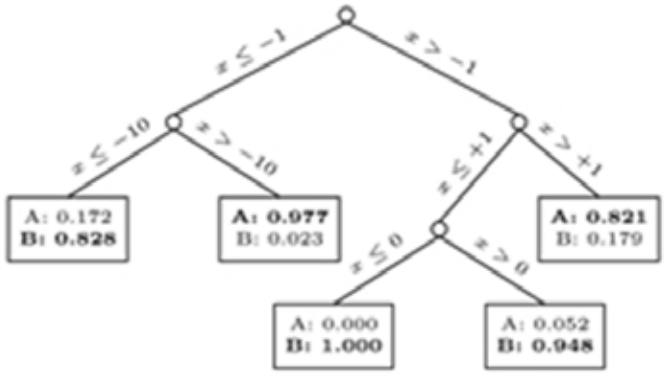


Fig. 2: Example Decision Tree Before Postpruning

Now, if we use the six tuples in Table 1 as test tuples4 and use this decision tree to classify them, we would classify tuples 2, 4, 6 as class "B" (the most likely class label in L), and hence, misclassify tuple 2. We would classify tuples 1, 3, 5 as class "A," thus getting the class label of 5 wrong. The accuracy is 2=3.

B. Distribution-Based Approach

For uncertain data, we adopt the same decision tree building framework as described above for handling point data. After an attribute Aj n and a split point zn have been chosen for a node n, we have to 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. Recall that the pdf of a tuple ti 2 S under attribute A_j n spans the interval ai;j; bi;j. If bi;jzn, the pdf of ti lies completely on the left n n n of the split point, and thus, ti is assigned to L. Similarly, we assign ti to R if zn < ai;j n. If the pdf properly contains the split point, i.e., ai;jzn < bi;j, we split ti into two fractional tuples tL and tR in the same way as described in Section 3.2 and add them to L and R, respectively. We call this algorithm Uncertain Decision Tree (UDT).

Again, the key to building a good decision tree is a good choice of an attribute Aj n and a split point zn 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. This is because a tuple ti 's pdf spans a continuous range ½ai;j ; bi;j . Moving

the split point froR ai;j to bi;jai;jn fi;jn ðxÞ dx (and likewise for pR). This changes the fractional tuples tL and tR , and thus, changes the resulting tree. If we model a pdf by s sample values, we are approximating the pdf by a discrete distribution of s points. In this case, as the split point moves from one end point ai;j to another end point bi;j of the interval, the probability pL changes in s steps. With m tuples, there are in total ms sample points. So, there are at most ms 1 possible split points to consider. Considering all k attributes, to determine the best (attribute, split point) pair thus requires us to examine kðms 1Þ combinations of attributes and split points. Comparing to AVG, UDT is s times more expensive.

Table 1: Selected Data Sets from the UCI Machine Learning Repository

Data Set	Training Tuples	No. of Attributes	No. of Classes	Test Tuples
JapaneseVowel	270	12	9	370
PenDigits	7494	16	10	3498
PageBlock	5473	10	5	10-fold
Satellite	4435	36	6	2000
Segment	2310	14	7	10-fold
Vehicle	846	18	4	10-fold
BreastCancer	569	30	2	10-fold
Ionosphere	351	32	2	10-fold
Glass	214	9	6	10-fold
Iris	150	4	3	10-fold

Note that splitting a tuple into two fractional tuples involves a calculation of the probability pL , which requires an integration. We remark that by storing the pdf in the form of a cumulative distribution, the integration can be done by simply subtracting two cumulative probabilities.

Let us reexamine the example tuples in Table 1 to see how the distribution-based algorithm can improve classification accuracy. By taking into account the probability distribution, UDT builds the tree shown in Fig. 3 before prepruning and postpruning are applied. This tree is much more elaborate than the tree shown in Fig. 2a because we are using more information, and hence, there are more choices of split points. The tree in Fig. 3 turns out to have a 100 percent classification accuracy. After postpruning, we get the tree in Fig. 2b. Now, let us use the six tuples in Table 1 as testing tuples to test the tree in Fig. 2b. For instance, the classification result of tuple 3 gives and PðBÞ ¼ 5=8 0:20 þ 3=8 0:788 ¼ 0:4205.

PðAÞ ¼ 5=8 0:80 þ 3=8 0:212 ¼ 0:5795

Since the probability for "A" is higher, we conclude that tuple 3

belongs to class "A." All the other tuples are handled similarly, using the label of the highest probability as the final classification result. It turns out that all six tuples are classified correctly. This handcrafted 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

To explore the potential of achieving a higher classification accuracy by considering data uncertainty, we have implemented AVG and UDT and applied them to 10 real data sets (see Table 2) taken from the UCI Machine Learning Repository [34]. These data sets are chosen because they contain mostly numerical attributes obtained from measurements. For the purpose of our experiments, classifiers are built on the numerical attributes and their "class label" attributes. Some data sets are already divided into "training" and "testing" tuples. For those that are not, we use 10-fold cross validation to measure the accuracy.

The first data set contains 640 tuples, each representing an utterance of the Japanese vowels by one of the nine participating male speakers. Each tuple contains 12 numerical attributes, which are Linear Predictive Coding (LPC) coefficients. These coefficients reflect important features of speech sound. Each attribute value consists of 7-29 samples of LPC coefficients collected over time.

Table 2: Accuracy Improvement by Considering the Distribution

Data Set	AVG	UDT							
		Best Case	Gaussian Distribution				Uniform Distribution		
			$w = 1\%$	$w = 5\%$	$w = 10\%$	$w = 20\%$	$w = 2\%$	$w = 10\%$	$w = 20\%$
JapaneseVowel	81.89	87.30	**87.30** (The distribution is based on samples from raw data)						
Pen-Digit	90.87	96.11	91.66	92.18	93.79	95.22	91.68	93.76	**96.11**
PageBlock	95.73	96.82	**96.82**	96.32	95.74	*94.87*	N/A		
Satellite	84.48	87.73	85.18	87.1	**87.73**	86.25	85.9	87.2	85.9
Segment	89.37	92.91	91.91	**92.91**	92.23	*89.11*	N/A		
Vehicle	71.03	75.09	72.44	72.98	73.18	**75.09**	*69.97*	71.04	71.62
BreastCancer	93.52	95.93	94.73	94.28	95.51	**95.93**	N/A		
Ionosphere	88.69	91.69	89.65	88.92	**91.69**	91.6	N/A		
Glass	66.49	72.75	69.6	**72.75**	70.79	*69.69*	N/A		
Iris	94.73	96.13	*94.47*	95.27	96	**96.13**	N/A		

These samples represent uncertain information and are used to model the pdf of the attribute for the tuple. The class label of each tuple is the speaker id. The classification task is to identify the speaker when given a test tuple. The other nine data sets contain "point values" without uncertainty. To control the uncertainty for sensitivity studies, we augment these data sets with uncertainty information generated as follows: We model uncertainty information by fitting appropriate error models on to the point data. For each tuple ti and for each attribute Aj , the point value vi;j reported in a data set is used as the mean of a pdf fi;j , defined over an interval ½ai;j ; bi;j . The range of values for Aj (over the whole data set) is noted and the width of ½ai;j ; bi;j is set to w jAjj, where jAjj denotes the width of the range for Aj and w is a controlled parameter.

To generate the pdf fi;j , we consider two options. The first is uniform distribution, which implies fi;jðxÞ ¼ ðbi;j ai;jÞ 1. The other option is Gaussian distribution5 for which we use 4 ðbi;jai;jÞ as the standard deviation. In both cases, the pdf is generated using s sample points in the interval. Using this method (with controllable parameters w and s, and a choice of Gaussian versus uniform distribution), we transform a data set with point values into one with uncertainty. The reason that we choose Gaussian distribution and uniform distribution is that most physical measures involve random noise, which follows Gaussian distribution, and that digitization of the measured values introduces quantization noise that is best described by a uniform distribution. Of course, most digitized measurements suffer from a combination of both kinds of uncertainties. For the purpose of illustration, we have only considered the two extremes of a wide spectrum of possibilities.

The results of applying AVG and UDT to the 10 data sets are shown in Table 3. As we have explained, under our uncertainty model, classification results are probabilistic. Following [3], we take the class label of the highest probability as the final class label. We have also run an experiment using C4.5 [3] with the information gain criterion. The resulting accuracies are very similar to those of AVG and are hence omitted. In the experiments, each pdf is represented by 100 sample points (i.e., s ¼ 100), except for the "JapaneseVowel" data set. We have repeated

the experiments using various values for w. For most of the data sets, Gaussian distribution is assumed as the error model. Since the data sets "PenDigits," "Vehicle," and "Satellite" have integer domains, we suspected that they are highly influenced by quantization noise. So, we have also tried uniform distribution on these three data sets, in addition to Gaussian.6 For the "JapaneseVowel" data set, we use the uncertainty given by the raw data (7-29 samples) to model the pdf.

From the table, we see that UDT builds more accurate decision trees than AVG does for different distributions over a wide range of w. For the first data set, whose pdf is modeled from the raw data samples, the accuracy is improved from 81.89 to 87.30 percent, i.e., the error rate is reduced from 18.11 to 12.70 percent, which is a very substantial improvement. Only in a few cases (marked by "# " in the table), does UDT give slightly worse accuracies than AVG. To better show the best potential improvement, we have identified the best cases (marked by " ") and repeated them in the third column of the table. Comparing the second and third columns of Table 3, we see that UDT can potentially build remarkably more accurate decision trees than AVG. For example, for the "Iris" data set, the accuracy improves from 94.73 to 96.13 percent.

(Thus, the error rate is reduced from 5.27 to 3.87 percent.) Using Gaussian distribution gives better accuracies in eight out of the nine data sets, where we have modeled the error distributions as described above. This suggests that the effects of random noise dominate quantization noise.

The exception is "PenDigits." As we have pointed out, this data set contains integral attributes, which is likely subject to quantization noise. By considering a uniform distribution as the error model, such a noise is taken into consideration, resulting in a high classification accuracy.

We have repeated the experiments and varied s, the number of sample points per pdf, from 50 to 200. There is no significant change in the accuracies as s varies. This is because we are keeping the pdf generation method unchanged. Increasing s improves our approximation to the distribution, but does not actually change the distribution.

This result, however, does not mean that it is unimportant to collect measurement values. In real applications, collecting more measurement values allows us to have more information to model the pdfs more accurately. This can help improve the quality of the pdf models. As shown above, modeling the probability distribution is very important when applying the distribution-based approach. So, it is still important to collect information on uncertainty. (We will discuss about this further in Section 7.1.)

D. Effect of Noise Model

How does the modeling of the noise affect the accuracy? In the previous section, we have seen that by modeling the error, we can build decision trees that are more accurate. It is natural to hypothesize that the closer we can model the error, the better will be the accuracy of the decision trees build by UDT.

V. Pruning Algorithms

Although UDT can build a more accurate decision tree, it is not as efficient as AVG. As we have explained, to determine the best attribute and split point for a node, UDT has to examine kðms 1Þ split points, where k ¼ number ofattributes, m ¼ number of tuples, and s ¼ number of samples per pdf. (AVG has to examine only kðm 1Þ split points.) For each such candidate attribute Aj and split point z, an entropy Hðz; AjÞ has to be computed (see (1)). Entropy calculations are the most computation-intensive part of UDT. Our approach to developing more efficient algorithms is to come up with strategies for pruning candidate split points and entropy calculations.

Note that we are considering safe pruning here. We are only pruning away candidate split points that give suboptimal entropy values.7 So, even after pruning, we are still finding optimal split points. Therefore, the pruning algorithms do not affect the resulting decision tree, which we have verified in our experiments. It only eliminates suboptimal candidates from consideration, thereby speeding up the tree building process.

A. Pruning Empty and Homogeneous Intervals

Recall that the BestSplit function in UDT is to solve the optimization problem of minimizing Hðz; AjÞ over all attributes Aj and all possible split points in domðAjÞ. Let us first focus on finding the best split point for one particular attribute Aj . (Note that there may be more than one best split point, each giving the same entropy value. Finding any one of them suffices.) We then repeat the process to find the best split point for every other attribute. The attribute with the best split point giving the lowest entropy is taken as the result of BestSplit.

We define the set of end points of tuples in S on attribute Aj as Qj ¼ fq j ðq ¼ ah;jÞ _ ðq ¼ bh;jÞ for some th 2 Sg. We assume that there are v such end points, q1 ; q2 ; . . . ; qv , sorted in ascending order. Within ½q1 ; qv , we want to find an optimal split point for attribute Aj .

Definition 1. For a given set of tuples S, an optimal split point for an attribute Aj is one that minimizes Hðz; AjÞ. (Note that the minimization is taken over all z 2 ½q1 ; qv .)

The end points define v1 disjoint intervals: ðqi ; qiþ1 for i ¼ 1; . . . ; v 1. We will examine each interval separately.

For convenience, an interval is denoted by ða; b .

DeRinition 2 (Empty interval). An interval ða; b is empty if ba f h;jðxÞ dx ¼ 0 for all th 2 S.

Definition 3 (Homogeneous interval). An interval ða; b is Romogeneous if there exists a class label c 2 C such that ba f h;jðxÞ dx ¼ 0) ch ¼ c for all th 2 S.

Intuitively, an interval is empty if no pdfs intersect it; an interval is homogeneous if all the pdfs that intersect it come from tuples of the same class.

Definition 4 (Heterogeneous interval). An interval ða; 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.

Proof. By the definition of information gain, if the optimal split point can be found in the interior of an empty interval ða; b , then that split point can be replaced by the end point a without changing the resulting entropy. u As a result of this theorem, if ða; b is empty, we only need to examine the end point a when looking for an optimal split point. There is a well-known analogue for the point data case, which states that if an optimal split point is to be placed between two consecutive attribute values, it can be placed anywhere in the interior of the interval and the entropy will be the same [28]. Therefore, when searching for the optimal split point, there is no need to examine the interior of empty intervals.

The following theorem further reduces the search space:

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.

Proof sPetch. Using the substitution x ¼ P c2C c;jða; zÞ and y ¼ (see Definition 6 below for the c;j function), the entropy Hðz; AjÞ can be rewritten in terms of x and y. It can be shown that the Hessian matrix 52 H is negative semidefinite. Therefore, Hðx; yÞ is a concave function, and hence, it attains its minimum value at the corners of the domain of ðx; yÞ, which is a convex polytope. It turns out that these corners correspond to z ¼ a or z ¼ b.

The implication of this theorem is that the interior points in 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. It states that if some consecutive attribute values come from tuples of the same class, then we do not need to consider splits with split points between those values [28].

Definition 5 (Tuple density). Given a class c 2 C, an attribute Aj , and a set of tuples S, we define the tuple density function gc;j as

$$g_{c;j} \;¼\; \sum_{\text{th } 2S:ch ¼c} wh\,fh;j\;;$$

where wh is the weight of the fractional tuple th 2 S (see Section 3.2).

This is a weighted sum of the pdf's fh;j of those tuples th 2 S whose class labels are c. With this function, we can define the tuple count of any class label within an interval:

Definition 6 (Tuple count). For an attribute Aj , the tuple count for class c 2 C in an interval ða; b is

$$c;jða; bÞ ¼ \int_a^b gc;jðxÞ\,dx:$$

The intention is that c;jða; bÞ gives the total number of tuples within the interval ða; b having a class label c, taking into account both the probability of occurrence of that class in the interval and the tuples' weights. Now, we are ready to state our next theorem, which is analogous to a similar result in [29] concerning data without uncertainty.

Theorem 3. Suppose that the tuple count for each class increases linearly in a heterogeneous interval ða; b (i.e., 8c 2 C; 8t 2 ½0; 1 ; c;jða; ð1tÞa þ tbÞ ¼ ct for some constant c). If an optimal split point falls in ða; b , then an end point of the interval is also an optimal split point.

Proof sketch. We use the substitution z ¼ ð1 tÞa þ tb to rewrite the entropy Hðz; AjÞ as a function of t. It can bedt20, and hence, HðtÞ is a concave function. Consequently, H attains its minimum value at one of the extreme points t ¼ 0 and t ¼ 1, which correspond to z ¼ a and z ¼ b, respectively. u One typical situation in which the condition holds is when all the pdfs follow the uniform distribution. In that case, the fraction of each tuple increases linearly in ða; b , therefore, the sum of a subset of them must also increase linearly. This has an important consequence: If all the pdfs are uniform distributions, then the optimal split point can be found among the 2jSj end points of the intervals of the tuples in S. Theorem 3, thus, reduces the number of split points to consider to OðjSjÞ.

In case the pdfs do not follow uniform distributions, Theorem 3 is not applicable. Then, we apply Theorems 1 and 2 to UDT to prune the interior points of empty and homogeneous intervals. This gives our Basic Pruning algorithm UDT-BP. Algorithm UDT-BP thus has to examine all end points of empty and homogeneous intervals as well as all sample points in heterogeneous intervals.

B. Pruning by Bounding

Our next algorithm attempts to prune away heterogeneous intervals through a bounding technique. First, we compute the entropy Hðq; AjÞ for all the end points q 2 Qj . Let Hj ¼ minq2Q j fHðq; AjÞg be the smallest of such end point entropy values. Next, for each heterogeneous interval ða; b , we compute a lower bound, Lj , of Hðz; AjÞ over all the candidate split points z 2 ða; b . If Lj Hj , we know that none of the candidate split points within the interval ða; b can give an entropy that is smaller than Hj , 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. So, the key to this pruning technique is to find a lower bound of Hðz; AjÞ that is not costly to compute, and yet is reasonably tight for the pruning to be effective. We have derived such a bound Lj given below. First, we introduce a few symbols to make the expression of the bound more compact and manageable:

$$nc ¼ c;jð 1; aÞ; mc ¼ c;jðb; þ1Þ; kc ¼ c;jða; bÞ;$$

$$n ¼ \sum_{c2C} nc\;;\; m ¼ \sum_{c2C} mc\;;\; N ¼ n þ$$

$$c ¼ nc þ kc$$

$$n þ kc\;;\; \text{and}\; c ¼ mc þ kcc :$$

$$\binom{}{kc} þ m;$$

Note that all these quantities are independent of the split point z. Our lower bound for Hðz; AjÞ is given by

$$Lj ¼ \frac{1}{N}\sum_{c2C} ½nc \log_2 c þ mc \log_2 c þ kc \log_2 ðmaxf c\;;\; cgÞ$$

We remark that the calculation of the lower bound is similar to entropy calculation. It thus costs about the same as the computation of a split point's entropy. 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 algorithm UDT-LP.

With UDT-LP, each attribute is processed independently: We determine a pruning threshold Hj for each attribute Aj to prune intervals in domðAjÞ. A better alternative is to compute a global threshold H ¼ min1 j kHj for pruning. In other words, we first compute the entropy values of all end points for all k attributes. The smallest such entropy is taken as the global pruning threshold H . This threshold is then used to prune heterogeneous intervals of all k attributes. We call this algorithm the Global Pruning algorithm UDT-GP.

C. End Point Sampling

As we will see later in Section 6, 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 the split points and the calculation of entropy-like lower bounds for intervals) to only 2.7 percent 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 note that the entropy Hðq; AjÞ of an end point q is computed for two reasons. First, for empty and homogeneous intervals, their end points are the only candidates for the optimal split point. Second, the minimum of all end point entropy values is used as a pruning threshold. For the latter purpose, we remark that it is unnecessary that we consider all end point entropy values. We can take a sample of the end points (say, 10 percent) 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. Also, we can concatenate a few consecutive intervals, say, I1 ; I2 , and I3 , into a bigger interval I, compute a lower bound for I based on (3), and attempt to prune I. If successful, we have effectively pruned the end points of I1 , I2 , and I3 .

We incorporate these end point Sampling strategies into UDT-GP. The resulting algorithm is called UDT-ES. We illustrate UDT-ES by an example shown in Fig. 5. (In this example, we ignore Theorems 1 and 2, concentrating on how end point sampling works.) The figure shows nine rows, illustrating nine steps of the pruning process. Each row shows an arrowed line representing the real number line. On this line, end points (represented by crosses) or

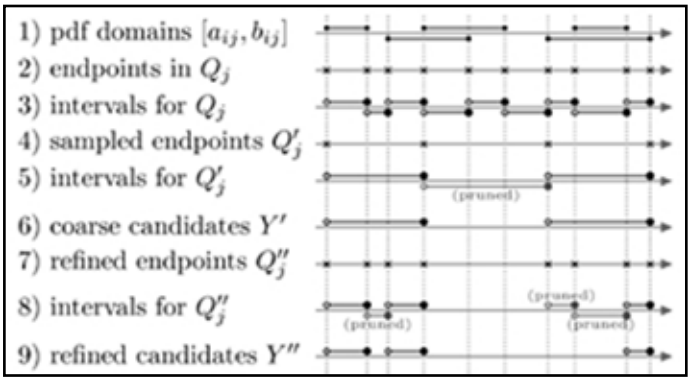


Fig. 3: Illustration of End Point Sampling

intervals (represented by line segments) are drawn. Row 1 shows the intervals obtained from the domains of the pdfs. The collection of end points of these intervals constitutes the set Qj (row 2). From these end points, disjoint intervals are derived (row 3). So far, the process is the same as global pruning. The next step differs from the global pruning algorithm: Instead of using the set of all end points Qj (row 2), we take a sample Q0 j (row 4) of these points. The choice of the sample size is a trade-off between fewer entropy calculations for the end points and a stronger pruning power. Our experiments have shown that 10 percent is a good choice of the end point sample size. Then, we continue with the global pruning algorithm as before, using the sampled end points Q0 j instead of Qj . The algorithm thus operates on the intervals derived from Q0 j (row 5) instead of those derived from Qj (row 3). Note that intervals in row 3 are concatenated to form intervals in row 5, and hence, fewer intervals and end points need to be processed. After all the prunings on the coarser intervals are done, we are left with a set Y 0 of candidate intervals (row 6). (Note that a couple of end points are pruned in the second interval of row 5.) For each unpruned candidate interval ðqy ; qyÞ1 inrow 6, we bring back the original set of end points inside the interval (row 7) and their original finer intervals (row 8). We reinvoke the global pruning again using the end points in Qj0 (carefully caching the already calculated values of Hðq; AjÞ for q 2 Q0 j). The candidate set of intervals obtained after pruning is Y 00 (row 9), which is a much smaller

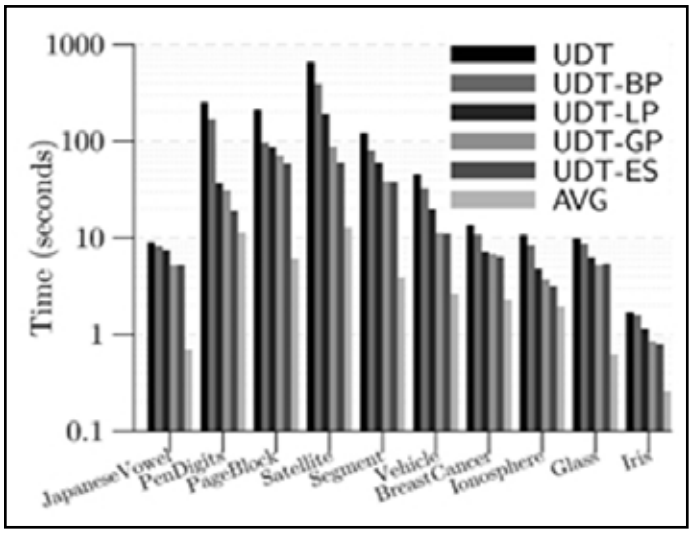


Fig. 4: Execution Time

candidate than the set of candidate intervals when no end point sampling is used. For the candidate intervals in Y 00 , we compute the values Hðz; AjÞ for all pdf sample points to find the minimum entropy value.

Experiments, to be presented in the next section, show that using end point sampling reduces a large number of entropy computations at the end points. It does lose some pruning effectiveness, but not significantly. Thus, the end point sampling pays off and improves the performance of the global pruning algorithm.

VI. Experiments on Efficiency

The algorithms described above have been implemented8 in Java using JDK 1.6 and a series of experiments was performed on a PC with an Intel Core 2 Duo 2.66 GHz CPU and 2 GB of main memory, running Linux kernel 2.6.22 i686. Experiments on the accuracy of our novel distribution-based UDT algorithm have been presented already in Section 4.2. In this section, we focus on the pruning effectiveness of our pruning algorithms and their runtime performance.

The data sets used are the same as those used in Section 4.2. The same method is used to synthesize data uncertainty. Only Gaussian distribution is used for the experiments below. We use the parameters s ¼ 100 (number of sample points per pdf) and w ¼ 10% (width of the pdf's domain, as a percentage of the width of the attribute's domain) as the baseline settings.

For the data set "JapaneseVowel," since its uncertainty is taken from raw data (7-29 samples per pdf), we cannot control its properties for sensitivity studies. So, it is excluded from Figs. 8 and 9. The bars for the "JapaneseVowel" in Figs. 6 and 7 are given for reference only.

A. Execution Time

We first examine the execution time of the algorithms, which is charted in Fig. 6. In this figure, six bars are drawn for each data set. The vertical axis, which is in log scale, represents the execution time in seconds. We have given also the execution time of the AVG algorithm (see Section 4.1). 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. In the figure, we observe the following general (ascending) order of efficiency: UDT, UDT-BP, UDT-LP, UDT-GP, and UDT-

ES. This agrees with the successive enhancements of these pruning techniques discussed in Section V.

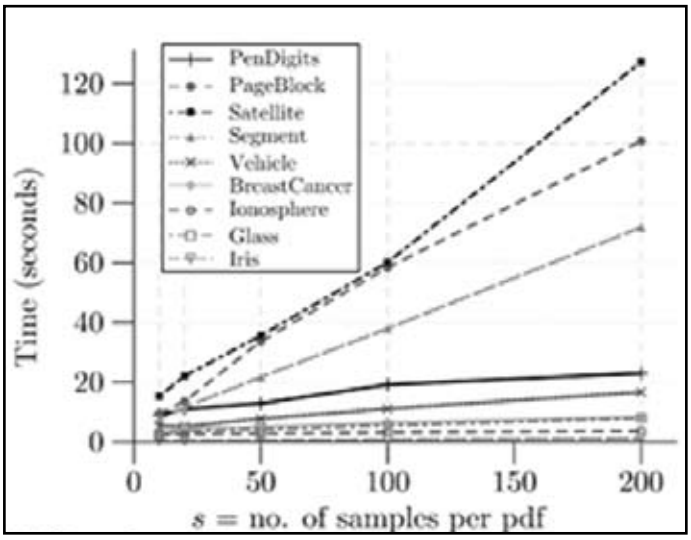


Fig. 5: Effects of s on UDT-ES

Minor fluctuations are expected as the pruning effectiveness depends on the actual distribution of the data. 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 4.3). Among the distribution-based algorithms, UDT-ES is the most efficient. It takes 62 percent ("Ionosphere") to 865 percent ("Segment") more time to finish than AVG. We remark that in the experiment, each pdf is represented by 100 sample points (i.e., s ¼ 100). Except for the data set "JapaneseVowel," all UDT-based algorithms thus have to handle 99 times more data than AVG, which only processes one average per pdf. For the data sets "PenDigits" and "Ionosphere," our pruning techniques are so effective that the execution time of UDT-ES is less than 1.7 times of that of AVG, while we achieve a much better classification accuracy (see Table 3). It worths to spend the extra time with the distribution-based algorithms for the higher accuracy of the resulting decision trees.

B. Pruning Effectiveness

In this section, we study the pruning effectiveness of the algorithms. Fig. 7 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. Note that Fig. 7 is also in log scale. The figure shows that our pruning techniques introduced in Section 5 are highly effective. Comparing the various bars against that for UDT, it is obvious that a lot of entropy calculations are avoided by our bounding techniques (see Section 5.2).

Indeed, UDT-BP only needs to perform 14-68 percent of the entropy calculations done by UDT. This corresponds to a pruning of 32-86 percent of the calculations. UDT-LP does even fewer calculations: only 5.4-54 percent of those of UDT. By using a global pruning threshold, UDT-GP only needs to compute 2.7-29 percent of entropy values compared with UDT. By pruning end points, UDT-ES further reduces the number of entropy calculations to 0.56-28 percent. . It thus achieves a pruning effectiveness ranging from 72 percent up to as much as 99.44 percent.

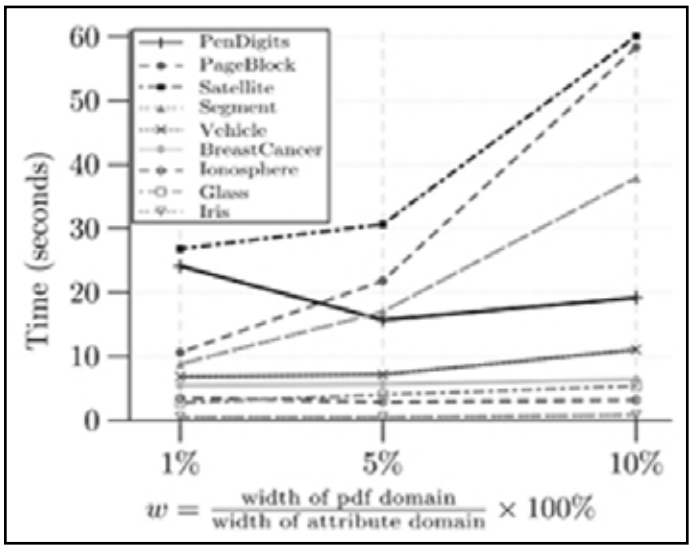


Fig. 6: Effects of w on UDT-ES.

As entropy calculations dominate the execution time of UDT, such effective pruning techniques significantly reduce the tree construction time.

C. Effects of s

To study the effects of the number of sample points per pdf (s) on the performance, we ran UDT-ES with different values of s. The results are shown in Fig. 8. The y-axis is in linear scale. For every data set, the execution time rises basically linearly with s. This is expected because with more sample points, the computations involved in the entropy calculation of each interval increase proportionately.

D. Effects of w

Another set of experiments was carried out to study the effects on the width of the pdf's domain as a percentage of the width of an attribute's domain (w). This parameter affects the distribution of the pdfs that are synthesized. In particular, the standard deviation chosen is a quarter of the width of the pdf's domain. Fig. 9 shows the execution times of the UDT-ES algorithm. The effect of w is different for different data sets. In general, a larger w causes the pdfs to span a wider range, increasing the chances that the pdf of one tuple overlaps with that of another tuple of a different class. Thus, there is a higher chance of getting heterogeneous intervals. Since UDT-ES spends most of its time on processing heterogeneous intervals, an increase in w causes UDT-ES to spend more time in general. This effect can vary from data set to data set, though, because the appearance of heterogeneous intervals depends very much on the distribution of data. For instance, in our experiments, the "PenDigits" data set does not follow the general trend.

Our experiments have thus shown that our novel algorithms, especially UDT-ES, are practical for a wide range of settings. We have experimented it with many real data sets with a wide range of parameters including the number of tuples, the number of attributes, and different application domains. We have also synthesized pdfs covering a wide range of error models, including Gaussian and uniform distribution and various widths (w) and granularities (s). Although UDT-ES inevitably takes more time than the classical approach AVG in building decision trees, it can potentially build more accurate trees because it takes the uncertainty information into account.

VII. Discussions

A. The Uncertainty Model

In our discussion, uncertainty models of attributes have been assumed known by some external means. In practice, finding a good model is an application-dependent endeavor. For example, manufacturers of some measuring instruments do specify in instruction manuals the error of the devices, which can be used as a source of information for modeling error distributions. In some other cases, repeated measurements can be taken and the resulting histogram can be used to approximate the pdf (as we have done in Section 4.3 with the "JapaneseVowel" data set). In the case of random noise, for example, one could fit a Gaussian distribution using the sample mean and variance, due to the Central Limit Theorem [35].

During the search for data sets appropriate for our experiments, we have hit a big obstacle: There are few data sets with complete uncertainty information. Although many data sets with numerical attributes have been collected via repeated measurements, very often the raw data have already been processed and replaced by aggregate values, such as the mean. The pdf information is thus not available to us. One example is the "BreastCancer" data set (see Table 2) from the UCI repository [34]. This data set actually contains 10 uncertain numerical features collected over an unspecified number of repeated measurements. However, when the data set is deposited into the repository, each of these 10 features is replaced by three attribute values, giving the mean, the standard score, and the mean of the three largest measured values. With these three aggregate values, we are unable to recover the distribution of each feature. Even modeling a Gaussian distribution is impossible: These three aggregate values are insufficient for us to estimate the variance. Had the people preparing this data set provided the raw measured values, we would be able to model the pdfs from these values directly, instead of injecting synthetic uncertainty and repeating this for different parameter values for w (see Section 4.4).

Now that we have established in this work that using uncertainty information modeled by pdfs can help us construct more accurate classifiers, it is highly advisable that data collectors preserve and provide complete raw data, instead of a few aggregate values, given that storage is nowadays very affordable.

B. Handling Categorical Attributes

We have been focusing on processing uncertain numerical attributes in this paper. How about uncertain categorical attributes? Like their numerical counterparts, uncertainty can arise in categorical attributes due to ambiguities, data staleness, and repeated measurements. For example, to cluster users based on access logs of HTTP proxy servers using (besides other attributes such as age) the top-level domain names (e.g., ".com," ".edu," ".org," ".jp,""".de," and ".ca") as an attribute, we obtain repeated "measurements" of this attribute from the multiple log entries generated by each user. The multiple values collected from these entries form a discrete distribution, which naturally describes the uncertainty embedded in this categorical attribute. The color of a traffic light signal, which is green at the time of recording, could have changed to yellow or even red in 5 seconds, withprobabilities following the programmed pattern of the signal. This is an example of uncertainty arising from data staleness. Colors of flowers recorded in a survey may divide human-visible colors into a number of categories, which may overlap with one another. Such ambiguities could be recorded as a distribution, e.g., 80 percent yellow and 20 percent pink. In all these cases, using a distribution to record the possible values (with corresponding probabilities) is a richer representation than merely recording the most likely value.

For a tuple ti with uncertain categorical attribute Aj, the value uncertainty can be modeled by a discrete probability distribution function fi;j : domðAjÞ ! ½0; 1 sa tisfying P x2domðAjÞ fi;jðxÞ ¼ 1. This is analogous to the case of uncertain numerical attribute. An internal node n in the decision tree corresponding to a categorical attribute Aj is not associated with a split point, though. Rather, n has many child nodes, each corresponding to a distinct value in domðAjÞ.9 The test to perform at node n is to check the value of Aj in the test tuple, and the action taken is to follow the branch to the child node corresponding to that attribute value.

To build a decision tree on uncertain data with a combination of numerical and categorical attributes, the same approach as described before can be followed: The tree is built recursively in a top-down manner, starting from the root. At each node, all the possible attributes (numerical or categorical) are considered. For each attribute, the entropy of the split is calculated and the attribute giving the highest information gain is selected. The node is assigned that attribute (and split point, if it is a numerical attribute) and the tuples are (fractionally) propagated to the child nodes. Each child node is then processed recursively. Toevaluate the entropy of a categorical attribute Aj , we (fractionally) split the tuples in question into a set of buckets fBvjv 2 domðAjÞg. Tuple tx is copied into Bv as a new tuple ty with weight wy ¼ fx;jðvÞ if and only if wy > 0. The pdfs of ty are inherited from tx , except for attribute Aj , which is set to fy;jðvÞ ¼ 1 and fy;jðwÞ ¼ 0 for all w ¼ v. The entropy for the split on Aj is calculated using all the buckets. As a heuristic, a categorical attribute that has already been chosen for splitting in an ancestor node of the tree need not be reconsidered because it will not give any information gain if the tuples in question are split on that categorical attribute again.

C. Handling Unbounded pdfs

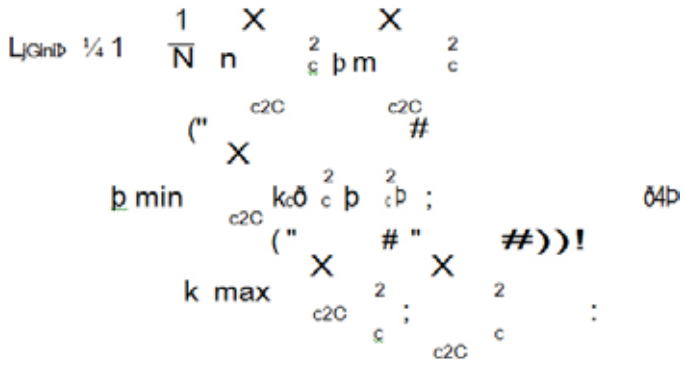
We have been assuming that the pdfs are bounded so that their end points (Qj) partition the real number line into a finite number of intervals for our pruning algorithms in Section 5 to work with. As suggested by the theorems in Section 5.1, there are good reasons to focus on the end points. For instance, when the pdfs are uniform, the end points are the only candidate split points that need to be considered.

However, in case the pdfs are unbounded, the pruning techniques can as well be applied to some artificial "end points." For example, suppose that we are handling attribute Aj . For each class c, we could treat the tuple count c;jð1; tÞ as a cumulative frequency function (of variable t) and select the 10-, 20-,. . . , 90-percentile points as the "end points." This generates 9 end points for each class, and 9jCj of them in total for all classes. These points can then be used with the UDT-GP and UDT-ES algorithms. The resulting intervals may not have the nice properties of the intervals defined by the real end points, such as the concavity of the entropy function. Yet it could still reduce the number of entropy computations. The actual effectiveness is subject to further research and experimentation.

D. Generalizing the Theorems

In Section 5.1, we have presented several theorems for pruning candidate split points when searching for an optimal split. We have assumed that entropy is used as the measure of dispersion. Indeed,

these theorems also hold when Gini index [30] is used as the dispersion measure. The proofs are similar and are omitted due to the space limitations. Consequently, the pruning techniques (Section 5) can be applied to Gini index as well. A different lower bound formula for Lj is needed, though:



Using this bound in the place of (3) and Gini index instead of entropy, we have repeated the experiments presented previously and got similar findings: UDT builds more accurate decision trees than AVG, and the pruning algorithms are highly effective, with UDT-ES being the most outstanding in performance.

Another popular dispersion measure used in the literature is gain ratio [3]. Unfortunately, we cannot prove Theorem 2 for gain ratio. This means that we can no longer prune away homogeneous intervals. Nevertheless, since Theorem 1 still holds, empty intervals can still be pruned away. Therefore, to handle gain ratio, we have to modify our pruning algorithms slightly: Empty intervals can still be pruned away as before; however, for both homogeneous and heterogeneous intervals, we have to apply the pruning by bounding technique.

E. Application to Point Data

While the techniques developed in this paper are mainly for the UDT algorithm for uncertain data, they can also be used to speed up the building of decision trees for point data. The techniques of pruning by bounding (Section 5.2) and end point sampling (Section 5.3) can be directly applied to point data to reduce the amount of entropy computations. The saving could be substantial when there are a large number of tuples.

VIII. Conclusions

We have found empirically that when suitable pdfs are used, exploiting data uncertainty leads to decision trees with remarkably higher accuracies. We therefore advocate that data be collected and stored with the pdf information intact. Performance is an issue, though, because of the increased amount of information to be processed, as well as the more complicated entropy computations involved.We have extended the model of decision tree classification to accommodate data tuples having numerical attributes with uncertainty described by arbitrary pdfs. We have modified classical decision tree building algorithms (based on the framework of C4.5 [3]) to build decision trees for classifying such data. Therefore, we have devised a series of pruning techniques to improve tree construction efficiency.Our algorithms have been experimentally verified to be highly effective. Their execution times are of an order of magnitude comparable to classical algorithms. Some of these pruning techniques are generalizations of analogous techniques for handling point-valued data. Other techniques, namely pruning by bounding and end point sampling, are novel. 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.

IX. Acknowledgments

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

References

[1] R. Agrawal, T. Imielinski, A.N. Swami,"Database Mining: A Performance Perspective", IEEE Trans. Knowledge and Data Eng., Vol. 5, No. 6, pp. 914-925, Dec. 1993.

[2] J.R. Quinlan,"Induction of Decision Trees", Machine Learning, Vol. 1, No. 1, pp. 81-106, 1986.

[3] J.R. Quinlan,"C4.5: Programs for Machine Learning. Morgan Kaufmann", 1993.

[4] C.L. Tsien, I.S. Kohane, 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, pp. 189-202, 2000.

[5] G.L. Freed , J.K. Fraley, "25 Percent "Error Rate" in Ear Temperature Sensing Device", Pediatrics, Vol. 87, No. 3, pp. 414-415, Mar. 1991.

[6] O. Wolfson, H. Yin, "Accuracy and Resource Consumption in Tracking and Location Prediction", Proc. Int'l Symp. Spatial and Temporal Databases (SSTD), pp. 325-343, July 2003.

[7] W. Street, W. Wolberg, O. Mangasarian,"Nuclear Feature Extraction for Breast Tumor Diagnosis", Proc. SPIE, pp. 861-870, [Online] Available: http://wwwciteseer.ist.psu.edu/street93nuclear.html, 1993.

[8] N.N. Dalvi, D. Suciu,"Efficient Query Evaluation on Probabilistic Databases", The VLDB J., Vol. 16, No. 4, pp. 523-544, 2007.

[9] E. Hung, L. Getoor, V.S. Subrahmanian, "Probabilistic Interval XML", ACM Trans. Computational Logic (TOCL), Vol. 8, No. 4, 2007.