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. Value uncertainty arises in many applications during the data collection process. Example sources of uncertainty include measurement/quantisation 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 utilised. We extend classical decision tree building algorithms to handle data tuples with uncertain values. Extensive experiments have been conducted that show that the resulting classifiers are more accurate than those using value averages. Since processing pdf's 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. To tackle this problem, we propose a series of pruning techniques that can greatly improve construction efficiency.**

***Index Terms*—Uncertain Data, Decision Tree, Classification, Data Mining**

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 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.

Before we delve into the details of our data model and algorithms, let us discuss the sources of data uncertainty and give some examples. Data uncertainty arises naturally in many applications due to various reasons. We briefly discuss three categories here: measurement errors, data staleness, and repeated measurements.

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 ear drum via an infrared sensor. A typical ear thermometer has a quoted *calibration error* of $\pm 0.2^{\circ}\mathrm{C}$, which is about 6.7% of the normal range of operation, noting that the human body temperature ranges from $37^{\circ}\mathrm{C}$ (normal) and to $40^{\circ}\mathrm{C}$ (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% of measurements are off by more than $0.5^{\circ}\mathrm{C}$, or about 17% of the operational range. Another source of error is quantisation errors introduced by the digitisation process. Such errors can be properly handled by assuming an appropriate error model, such as a Gaussian error distribution for random noise or a uniform error distribution for quantisation errors.

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 whereabout of a mobile device can only be approximated by imposing an uncertainty model on its last reported location[6]. 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

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

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. When we inquire about a patient's temperature, or wind speed, or the temperature of a certain section of the shuttle, which values shall we use? Or, would it be better to utilise all the information by considering the distribution given by the collected data values?

As a more elaborate example, consider the "BreastCancer" dataset reported in [7]. This dataset 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. From (1) and (2), we see that a *radius* is computed from a large number of measurements with a wide range of values. The source data points thus form interesting distributions. From (3), the fuzziness of the 2D image can be modelled by allowing a radius measure be represented by a range instead of a concrete point-value.

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 respondent is not so sure about the answer himself. In this example, the survey can restrict an answer to fall into a few pre-set 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:

1) A basic algorithm for constructing decision trees out of uncertain datasets.
2) A study comparing the classification accuracy achieved by the Averaging approach and the Distribution-based approach.
3) A set of mathematical theorems that allow 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.
5) A performance analysis on the various algorithms through a set of experiments.

In the rest of this 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 modelled appropriately. Pruning techniques to improve our new algorithm are devised in Section V, and experimental studies on the performance are presented in Section VI. Finally, we briefly discuss some related problems for further investigation in Section VII and conclude the paper in Section VIII.

II. Related Works

There has been significant research interest in uncertain data management in recent years. Data uncertainty has been broadly classified as 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[8]. "Probabilistic databases" have been applied to semi-structured data and XML[9], [10]. 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[11], [12]. 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. For example, indexing solutions for range queries on uncertain data[13], solutions for aggregate queries[14] such as nearest neighbour queries, and solutions for imprecise location-dependent queries[11] have been proposed.

There has been a growing interest in uncertain data mining. In [12], the well-known k-means clustering algorithm is extended to the UK-means algorithm for clustering uncertain data. As we have explained, data uncertainty is usually captured by pdf's, which are generally represented by sets of sample values. Mining uncertain data is therefore computationally costly due to information explosion (sets of samples vs. single values). To improve the performance of UK-means, pruning techniques have been proposed. Examples include min-max-dist pruning[15] and CK-means[16]. Apart from studies in partition-based uncertain data clustering, other directions in uncertain data mining include density-based clustering (e.g., FDBSCAN[17]), frequent itemset mining[18] and density-based classification[19]. Density-based classification requires that the joint probability distribution of the data attributes be known. In [19], each data point is given an error model. Upon testing, each test tuple is a point-valued data. These are very different from our data model, as we do not require the knowledge of the joint probability distribution of the data attributes. Each attribute is handled independently and may have its own error model. Further, the test tuples, like the training tuples, may contain uncertainty in our model.

Decision tree classification on uncertain data has been addressed for decades in the form of missing values[2], [3].

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[20] and probabilistic attribute tree[21]). In C4.5[3] and probabilistic decision trees[22], 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. We have also adopted the idea of probabilistic classification results. We do not directly address the problem of handling missing values. Rather, we tackle the problem of handling data uncertainty in a more general form. Our techniques are general enough for the existing missing-value handling methods to be encapsulated naturally into our framework. Based on the previously described approaches, a simple method of "filling in" the missing values could be adopted to handle the missing values, taking advantage of the capability of handling arbitrary pdf's in our approach. We can take the average of the pdf of the attribute in question over the tuples where the value is present. The result is a pdf, which can be used as a "guess" distribution of the attribute's value in the missing tuples. Then, we can proceed with decision tree construction.

Another related topic is fuzzy decision tree. Fuzzy information models data uncertainty arising from human perception and understanding[23]. 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[23]. 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[24], [25] and Soft Decision Tree[26]. In these models, a node of the decision tree does not give a crisp test which decides deterministically 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. Based on the fuzzy truth value of the test, the tuple is split into weighted tuples (akin to fractional tuples) and these are sent down the tree in parallel. This differs from the approach taken in this paper, in which the probabilistic part stems from the uncertainty embedded in the data tuples, while the test represented by each node of our decision tree remains crisp and deterministic. The 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 [27]. A numerical attribute usually has a possibly infinite domain of real or integral numbers, inducing a large search space for the best "split point". Given a set of n training tuples with a numerical attribute, there are as many as $n - 1$ binary split points or ways to partition the set of tuples into two non-empty groups. Finding the best split point is thus computationally expensive. To improve efficiency, many techniques have been proposed to reduce the number of candidate split points[28], [27], [29]. These techniques utilise the convex property of well-known evaluation functions like Information Gain[2] and Gini Index[30]. For the evaluation function TSE (Training Set Error), which is convex but not strictly convex, one only needs to consider the "alternation points" as candidate split points.[31] An alternation point is a point at which the ranking of the classes (according to frequency) changes. In this paper, we consider only strictly convex evaluation functions. (See Section VII-D for a brief discussion on how non-convex functions can be handled.) Compared to those works, ours can be considered an extension of their optimisation techniques for handling uncertain data (see Section V-A). In addition, we have introduced novel pruning techniques that could be applicable in handling point-valued data when the number of data tuples is huge (see Sections V-B, V-C and VII-E).

III. Problem Definition

This section formally defines the problem of decision-tree classification on uncertain data. We first discuss traditional decision trees briefly. Then, we discuss how data tuples with uncertainty are handled.

A. Traditional Decision Trees

In our model, a dataset consists of d *training* tuples, $\{t_1, t_2, \ldots, t_d\}$, and k numerical (real-valued) feature attributes, $A_1, \ldots A_k$. The domain of attribute A_j is $\mathrm{dom}(A_j)$. Each tuple t_i is associated with a feature vector $V_i = (v_{i,1}, v_{i,2}, \ldots, v_{i,k})$ and a class label c_i, where $v_{i,j} \in \mathrm{dom}(A_j)$ and $c_i \in C$, the set of all class labels. The classification problem is to construct a model M that maps each feature vector $(v_{x,1}, \ldots, v_{x,k})$ to a probability distribution P_x on C such that given a *test* tuple $t_0 = (v_{0,1}, \ldots, v_{0,k}, c_0)$, $P_0 = M(v_{0,1}, \ldots, v_{0,k})$ predicts the class label c_0 with high accuracy. We say that P_0 predicts c_0 if $c_0 = \arg\max_{c \in C} P_0(c)$.

In this paper we study *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 $z_n \in \mathrm{dom}(A_{j_n})$, giving a binary test $v_{0,j_n} \leq z_n$. An internal node has exactly 2 children, which are labelled "left" and "right", respectively. Each leaf node m in the decision tree is associated with a discrete probability distribution P_m over C. For each $c \in C$, $P_m(c)$ 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 $t_0 = (v_{0,1}, \ldots, v_{0,k}, ?)$, 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 $v_{0,j_n} \leq z_n$ and proceed to the left child or the right child accordingly. Eventually, we reach a leaf node m. The probability distribution P_m associated with m gives the probabilities that t_0 belongs to each class label $c \in C$. For a single result, we return the class label $c \in C$ that maximises $P_m(c)$.

B. Handling Uncertainty Information

Under our uncertainty model, a feature value is represented not by a single value, $v_{i,j}$, but by a pdf, $f_{i,j}$. For practical reasons, we assume that $f_{i,j}$ is non-zero only within a bounded interval $[a_{i,j}, b_{i,j}]$. (We will briefly discuss how our methods can be extended to handle pdf's with unbounded domains in Section VII-C.) A pdf $f_{i,j}$ 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 $x \in [a_{i,j}, b_{i,j}]$ with the associated value $f_{i,j}(x)$, effectively approximating $f_{i,j}$ by a discrete distribution with s possible values. 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. On the down side, processing large numbers 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 t_0 contains uncertain attributes. Its feature vector is thus a vector of pdf's $(f_{0,1}, \ldots, f_{0,k})$. 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 t_x with a weight $w_x \in [0, 1]$. Further, we recursively define the quantity $\phi_n(c; t_x, w_x)$, which can be interpreted as the conditional probability that t_x has class label c, when the subtree rooted at n is used as an uncertain decision tree to classify tuple t_x with weight w_x.

For each internal node n (including the root node), to determine $\phi_n(c; t_x, w_x)$, we first check the attribute A_{j_n} and split point z_n of node n. Since the pdf of t_x under attribute A_{j_n} spans the interval $[a_{x,j_n}, b_{x,j_n}]$, we compute the "left" probability $p_L = \int_{a_{x,j_n}}^{z_n} f_{x,j_n}(t)\,dt$ (or $p_L = 0$ in case $z_n < a_{x,j_n}$) and the "right" probability $p_R = 1 - p_L$. Then, we split t_x into 2 fractional tuples t_L and t_R. (The concept of fractional tuples is also used in C4.5[3] for handling missing values.) Tuples t_L and t_R inherit the class label of t_x as well as the pdf's of t_x for all attributes except A_{j_n}. Tuple t_L is assigned a weight of $w_L = w_x \cdot p_L$ and its pdf for A_{j_n} is given by

$$f_{L,j_n}(x) = \begin{cases} f_{x,j_n}(x)/w_L & \text{if } x \in [a_{x,j_n}, z_n] \\ 0 & \text{otherwise} \end{cases}$$

Tuple t_R is assigned a weight and pdf analogously. We define $\phi_n(c; t_x, w_x) = p_L \cdot \phi_{n_L}(c; t_L, w_L) + p_R \cdot \phi_{n_R}(c; t_R, w_R)$ where n_L and n_R 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 P_m over C. We define $\phi_m(c; t_x, w_x) = w_x \cdot P_m(c)$. Finally, for each class c, let $P(c) = \phi_r(c; t_0, 1.0)$, where r is the root node of the decision tree. Obtained this way, each probability $P(c)$ indicates how likely it is that the *test* tuple t_0 has class label c. These computations are illustrated in Figure 1, which shows a test tuple t_0 with one feature whose pdf has the domain $[-2.5, 2]$. 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 $p_L = 0.3$ and $p_R = 0.7$. So, t_0 is split into two tuples t_L and t_R with weights $w_L = 0.3$ and $w_R = 0.7$. The tuple t_L inherits the pdf from t_0 over the sub-domain $[-2.5, -1]$, normalised by multiplying by a factor of $1/w_L$. Tuple t_R inherits the pdf from t_0 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 t_0, giving $P(A) = 0.59$; $P(B) = 0.41$.

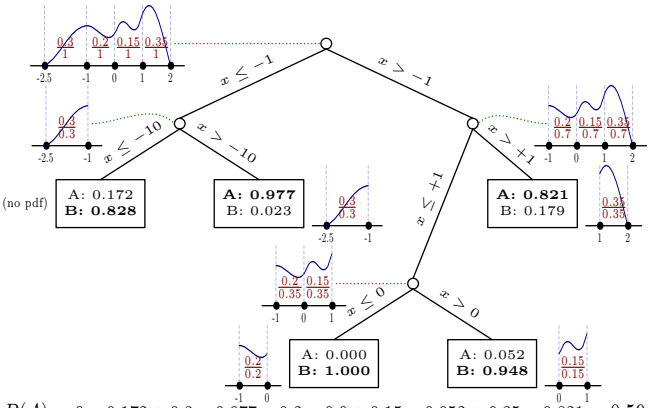


Fig. 1. Classifying a test tuple

If a single class label is desired as the result, we select the class label with the highest probability as the final answer. In the example in Figure 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 A_{j_n} and a good split point z_n for each internal node n, as well as an appropriate probability distribution P_m 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 dataset to a point-valued one by replacing each pdf with its mean value. More specifically, for each tuple t_i and attribute A_j, we take the mean value[1] $v_{i,j} = \int_{a_{i,j}}^{b_{i,j}} x f_{i,j}(x)\,dx$ as its representative value. The feature vector of t_i is thus transformed to $(v_{i,1}, \ldots, v_{i,k})$. A decision tree can then be built by applying a traditional tree construction algorithm.

To exploit the full information carried by the pdf's, our second approach, called "Distribution-based", considers all the sample points that constitute each pdf. The challenge here is

[1] One may alternatively use median or other summary statistics.

that a training tuple can now "pass" a test at a tree node *probabilistically* when its pdf properly contains the split point of the test. Also, a slight change of the split point modifies that probability, potentially altering the tree structure. We present details of the tree-construction algorithms under the two approaches in the following subsections.

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 and C4.5[3] can be reused. We call this approach AVG (for Averaging). We use an algorithm based on C4.5. Here is a brief description.

AVG is a greedy algorithm that builds a tree top-down. When processing a node, we examine a set of tuples S. The algorithm starts with the root node and with S being the set of all training tuples. At each node n, we first check if all the tuples in S have the same class label c. If so, we make n a leaf node and set $P_n(c) = 1$, $P_n(c') = 0\ \forall c' \neq c$. Otherwise, we select an attribute A_{j_n} and a split point z_n and divide the tuples into two subsets: "left" and "right". All tuples with $v_{i,j_n} \leq z_n$ are put in the "left" subset L; the rest go to the "right" subset R. If either L or R is empty (even after exhausting all possible choices of A_{j_n} and z_n), it is impossible to use the available attributes to further discern the tuples in S. In that case, we make n a leaf node. Moreover, the population of the tuples in S for each class label induces the probability distribution P_n. In particular, for each class label $c \in C$, we assign to $P_n(c)$ the fraction of tuples in S that are labelled c. If neither L nor R is empty, we make n an internal node and create child nodes for it. We recursively invoke the algorithm on the "left" child and the "right" child, passing to them the sets L and R, respectively.

To build a good decision tree, the choice of A_{j_n} and z_n is crucial. At this point, we may assume that this selection is performed by a blackbox algorithm BestSplit, which takes a set of tuples as parameter, and returns the best choice of attribute and split point for those tuples. We will examine this blackbox in details. Typically, BestSplit is designed to select the attribute and split point that minimises the degree of dispersion. The degree of dispersion can be measured in many ways, such as entropy (from information theory) or Gini index[30]. The choice of dispersion function affects the structure of the resulting decision tree.[2] In this paper we assume that entropy is used as the measure since it is predominantly used for building decision trees. (Our methods are also valid for Gini index. See Section VII-D.) The minimisation is taken over the set of all possible attributes A_j ($j = 1, \ldots, k$), considering all possible split points in dom(A_j). Given a set $S = \{t_1, \ldots, t_m\}$ of m tuples with point values, there are only $m-1$ ways to partition S into two non-empty L and R sets. For each attribute A_j, the split points to consider are given by the set of values of the tuples under attribute A_j, i.e., $\{v_{1,j}, \ldots v_{m,j}\}$. Among these values, all but the largest one give valid split points. (The largest one gives an empty R set, so invalid.)

For each of the $(m-1)k$ combinations of attributes (A_j) and split points (z), we divide the set S into the "left" and "right" subsets L and R. We then compute the entropy for each such combination:

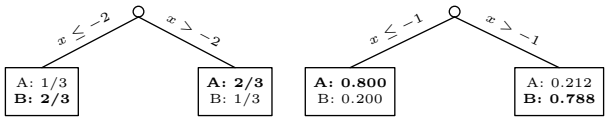
$$H(z, A_j) = \sum_{X=L,R} \frac{|X|}{|S|} \left(\sum_{c \in C} -p_{c/X} \log_2 p_{c/X} \right) \quad (1)$$

where $p_{c/X}$ is the fraction of tuples in X that are labelled c. We take the pair of attribute A_{j^*} and split point z^* that minimises $H(z, A_j)$ and assign to node n the attribute A_{j^*} with split point z^*.[3]

Let us illustrate this classification algorithm using the example tuples shown in Table I. This set consists of 6 tuples of 2 class labels "A" and "B". Each tuple has only 1 attribute, whose (discrete) probability distribution is shown under the column "probability distribution". For instance, tuple 3 has class label "A" and its attribute takes the values of -1, $+1$, $+10$ with probabilities $5/8$, $1/8$, $2/8$ respectively. The column "mean" shows the expected value of the attribute. For example, tuple 3 has an expected value of $+2.0$. With Averaging, there is only 1 way to partition the set: the even numbered tuples go to L and the odd-numbered tuples go to R. The tuples in each subset have the same mean attribute value, and hence cannot be discerned further. The resulting decision tree is shown in Figure 2(a). Since the left subset has 2 tuples of class B and 1 tuple of class A, the left leaf node L has the probability distribution $P_L(A) = 1/3$ and $P_L(B) = 2/3$ over the class labels. The probability distribution of class labels in the right leaf node R is determined analogously. Now, if we use the 6 tuples in Table I as test tuples[4] and use this decision tree to

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	



(a) Averaging (b) Distribution-based

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

[2] Analysis in [32] has discovered that using Gini index tends to put tuples of the majority class into one subset and the remaining tuples into the other subset. Entropy, on the other hand, prefers to balance the sizes of the resulting subsets.

[3] To alleviate the problem of over-fitting, we apply the techniques of *pre-pruning* and *post-pruning* (see [33], [3] for details).

[4] In practice and in the following experiments, *disjoint* training sets and testing sets are used. In this *hand-crafted* example, however, we use the same tuples for both training and testing just for illustration.

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

For uncertain data, we adopt the same decision tree building framework as described above for handling point data. After an attribute A_{j_n} and a split point z_n has 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 $t_i \in S$ under attribute A_{j_n} spans the interval $[a_{i,j_n}, b_{i,j_n}]$. If $b_{i,j_n} \leq z_n$, the pdf of t_i lies completely on the left of the split point and thus t_i is assigned to L. Similarly, we assign t_i to R if $z_n < a_{i,j_n}$. 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* t_L and t_R in the same way as described in Section III-B and add them to L and R, respectively. We call this algorithm UDT (for Uncertain Decision Tree).

Again, 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. This is because a tuple t_i's pdf spans a continuous range $[a_{i,j}, b_{i,j}]$. Moving the split point from $a_{i,j}$ to $b_{i,j}$ continuously changes the probability $p_L = \int_{a_{i,j_n}}^{z_n} f_{i,j_n}(x)\,dx$ (and likewise for p_R). This changes the fractional tuples t_L and t_R, 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 $a_{i,j}$ to another end-point $b_{i,j}$ of the interval, the probability p_L 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 require us to examine $k(ms-1)$ combinations of attributes and split points. Comparing to AVG, UDT is s time more expensive.

Note that splitting a tuple into two fractional tuples involves a calculation of the probability p_L, 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 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 is much more elaborate than the tree shown in Figure 2(a), because we are using more information and hence there are more choices of split points. The tree in Figure 3 turns out to have a 100% classification accuracy! After post-pruning, we get the tree in Figure 2(b). Now, let us use the 6 tuples in Table I as testing tuples[4] to test the tree in Figure 2(b). For instance, the classification result of tuple 3 gives $P(A) = 5/8 \times 0.80 + 3/8 \times 0.212 = 0.5795$ and $P(B) = 5/8 \times 0.20 + 3/8 \times 0.788 = 0.4205$. 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 6 tuples are classified correctly. This hand-crafted example thus illustrates that by considering probability distributions rather than just expected values, we can potentially build a more accurate decision tree.

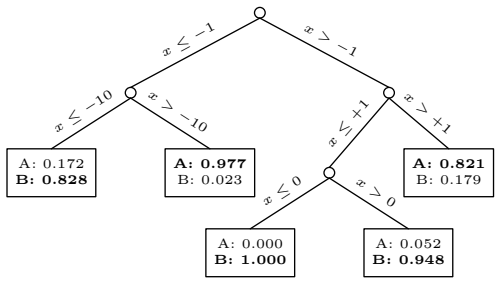


Fig. 3. Example decision tree before post-pruning

TABLE II
SELECTED DATASETS 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

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 II) taken from the UCI Machine Learning Repository[34]. These datasets 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 Japanese vowels by one of the 9 participating male speakers. Each tuple contains 12 numerical attributes, which are LPC (Linear Predictive Coding) coefficients. These coefficients reflect important features of speech sound. Each attribute value consists of 7–29 samples of LPC coefficients collected over time. 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 9 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 t_i and for each attribute A_j, the point value $v_{i,j}$ reported in a dataset is used as the mean of a pdf $f_{i,j}$, defined over an interval $[a_{i,j}, b_{i,j}]$. The range of values for A_j (over the whole data set) is noted and the width of $[a_{i,j}, b_{i,j}]$ is set to $w \cdot |A_j|$, where $|A_j|$ denotes the width of the range for A_j and w is a controlled parameter. To generate the pdf $f_{i,j}$, we consider two options. The first is uniform distribution, which implies $f_{i,j}(x) = (b_{i,j} - a_{i,j})^{-1}$. The other option is Gaussian distribution[5], for which we use $\frac{1}{4}(b_{i,j} - a_{i,j})$ 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 vs. 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 digitisation of the measured values introduces quantisation noise that is best described by a uniform distribution. Of course, most digitised 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 datasets are shown in Table III. 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., $s = 100$), except for the "JapaneseVowel" data set. We have repeated the experiments using various values for w. For most of the datasets, 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 quantisation noise. So, we have also tried uniform distribution on these three datasets, 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 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 substantial improvement. Only in a few cases (marked with "$_{\#}$" 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 with "*") and repeated them in the third column of the table. Comparing the second and third columns of Table III, 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%. (Thus, the error rate is reduced from 5.27% down to 3.87%.)

TABLE III
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		

Using Gaussian distribution gives better accuracies in 8 out of the 9 datasets where we have modelled the error distributions as described above. This suggests that the effects of random noise dominates quantisation noise. The exception is "PenDigits". As we have pointed out, this dataset contains integral attributes, which is likely subject to quantisation noise. By considering a uniform distribution as the error model, such 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 allow us to have more information to model the pdf's more accurately. This can help improve the quality of the pdf models. As shown above, modelling 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 VII-A.)

D. Effect of Noise Model

How does the modelling of the noise affect the accuracy? In the previous section, we have seen that by modelling the error, we can build decision trees which are more accurate. It is natural to hypothesise that the closer we can model the error, the better the accuracy will be. We have designed the following experiment to verify this claim.

In the experiment above, we have taken data from the UCI repository and directly added uncertainty to it so as to test our UDT algorithm. The amount of errors in the data is uncontrolled. So, in the next experiment, we inject some artificial noise into the data in a controlled way. For each dataset (except "JapaneseVowel"), we first take the data from the repository. For each tuple t_i and for each attribute A_j, the point value $v_{i,j}$ is perturbed by adding a Gaussian noise with

[5] Strictly speaking, Gaussian distribution has non-zero density on the whole real-number line. Here, we assume the Gaussian distribution is chopped at both ends symmetrically, and the remaining non-zero region around the mean is renormalised.

[6] For the other datasets, we did not use uniform distribution, and we indicate this with "N/A" (meaning "not applicable") in the table.

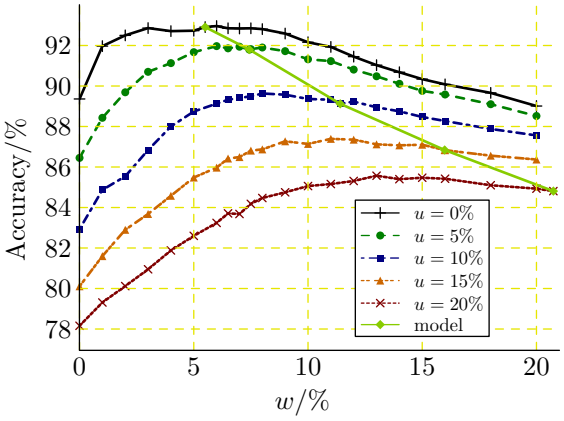


Fig. 4. Experiment with controlled noise on dataset "Segment"

zero mean and a standard deviation equal to $\sigma = \frac{1}{4}(u \cdot |A_j|)$, where u is a controllable parameter. So, the perturbed value is $\tilde{v}_{i,j} = v_{i,j} + \delta_{i,j}$, where $\delta_{i,j}$ is a random number which follows $N\left(0, \sigma^2\right)$. Finally, on top of this perturbed data, we add uncertainty as described in the last subsection (with parameters w and s). Then, we run AVG and UDT to obtain the accuracy, for various values u and w (keeping $s = 100$).

If the data from the UCI repository were ideal and 100% error-free, then we should expect to get the highest accuracy whenever $u = w$. This is because when $u = w$, the uncertainty that we use would perfectly model the perturbation that we have introduced. So, if our hypothesis is correct, UDT would build decision trees with the higher accuracy. Nevertheless, we cannot assume that the datasets are error-free. It is likely that some sort of measurement error is incurred in the collection process of these datasets. For simplicity of analysis, let us assume that the errors are random errors $\epsilon_{i,j}$ following a Gaussian distribution with zero mean and some unknown variance $\check{\sigma}^2$. So, if $\check{v}_{i,j}$ is the true, noise-free (but unknown to us) value for attribute A_j in tuple t_i, then $v_{i,j} = \check{v}_{i,j} + \epsilon_{i,j}$. Hence, in the perturbed dataset, we have $\tilde{v}_{i,j} = \check{v}_{i,j} + \epsilon_{i,j} + \delta_{i,j}$. Note that the total amount of noise in this value is $\epsilon_{i,j} + \delta_{i,j}$, which is the sum of two Gaussian-distributed random variables. So, the sum itself also follows the Gaussian distribution $N(0, \tilde{\sigma}^2)$ where $\tilde{\sigma}^2 = \check{\sigma}^2 + \sigma^2$. If our hypothesis is correct, then UDT should give the best accuracy when w matches this $\tilde{\sigma}$, i.e., when $\left(\frac{w \cdot |A_j|}{4}\right)^2 = \tilde{\sigma}^2 = \check{\sigma}^2 + \sigma^2 = \check{\sigma}^2 + \left(\frac{u \cdot |A_j|}{4}\right)^2$, i.e.,

$$w^2 = (\kappa \cdot \check{\sigma})^2 + u^2 \tag{2}$$

for some constant κ.

We have carried out this experiment and the results for the dataset "Segment" is plotted in Figure 4. Each curve (except the one labelled "model", explained below) in the diagram corresponds to one value of u, which controls the amount of perturbation that has been artificially added to the UCI dataset. The x-axis corresponds to different values of w—the error model that we use as the uncertainty information. The y-axis gives the accuracy of the decision tree built by UDT. Note that for points with $w = 0$, the decision trees are built by AVG. To interpret this figure, let us first examine each curve (i.e., when u is fixed). It is obvious from each curve that UDT gives significantly higher accuracies than AVG. Furthermore, as w increases from 0, the accuracy rises quickly to up a plateau, where it remains high (despite minor fluctuations). This is because as w approaches the value given by (2), the uncertainty information models the error better and better, yielding more and more accurate decision trees. When w continues to increase, the curves eventually drop gradually, showing that as the uncertainty model deviates from the error, the accuracy drops. From these observations, we can conclude that UDT can build significantly more accurate decision trees than AVG. Moreover, such high accuracy can be achieved over a wide range of w (along the plateaux). So, there is a wide error-margin for estimating a good w to use.

Next, let us compare the different curves, which correspond to different values of u—the artificially controlled perturbation. The trend observed is that as u increases, accuracy decreases. This agrees with intuition: the greater the degree of perturbation, the more severely are the data contaminated with noise. So, the resulting decision trees become less and less accurate. Nevertheless, with the help of error models and UDT, we are still able to build decision trees of much higher accuracies than AVG.

Last but not least, let us see if our hypothesis can be verified. Does a value of w close to that given by (2) yield a high accuracy? To answer this question, we need to estimate the value of $\kappa \cdot \check{\sigma}$. We do this by examining the curve for $u = 0$. Intuitively, the point with the highest w should give a good estimation of $\kappa \cdot \check{\sigma}$. However, since the curve has a wide plateau, it is not easy to find a single value of w to estimate the value. We adopt the following approach: Along this curve, we use the accuracy values measured from the repeated trials in the experiment to estimate a 95%-confidence interval for each data point, and then find out the set of points whose confidence interval overlaps with that of the point of highest accuracy. This set of points then gives a range of values of w, within which the accuracy is high. We take the mid-point of this range as the estimate for $\kappa \cdot \check{\sigma}$. Based on this value and (2), we calculate a value of w for each u, and repeat the experiments with each such pair of (u, w). The accuracy is measured and plotted in the same figure as the curve labelled "model". From the figure, it can be seen that the points where the "model" curve intersects the other curves lie within the plateau of them. This confirms our claim that when the uncertain information models the error closely, we can get high accuracies.

We have repeated this experiment with all the other datasets shown in Table II (except "JapaneseVowel"), and the observations are similar. Thus, we conclude that our hypothesis is confirmed: The closer we can model the error, the better will be the accuracy of the 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 $k(ms - 1)$ split points, where k = number of attributes, m = number of tuples, and s = number of samples per pdf. (AVG has to examine only $k(m - 1)$ split

points.) For each such candidate attribute A_j and split point z, an entropy $H(z, A_j)$ 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 sub-optimal 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 sub-optimal 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 optimisation problem of minimising $H(z, A_j)$ over all attributes A_j and all possible split points in $\mathrm{dom}(A_j)$. Let us first focus on finding the best split point for one particular attribute A_j. (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 A_j as $Q_j = \{q \mid (q = a_{h,j}) \vee (q = b_{h,j})$ for some $t_h \in S\}$. We assume that there are v such end-points, $q_1, q_2, \ldots, q_v$, 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 $H(z, A_j)$. (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 $i = 1, \ldots, v - 1$. 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 $\int_a^b f_{h,j}(x)\,dx \neq 0 \Rightarrow c_h = c$ for all $t_h \in S$.

Intuitively, an interval is empty if no pdf's intersect 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.

Proof: By the definition of information gain, if the optimal split point can be found in the interior of an empty interval $(a, b]$, then that split point can be replaced by the end-point a without changing the resulting entropy. ∎

[7] We may prune away some optimal split points (as in the case of uniform distribution), but we only do so when we are sure that at least one other optimal split point (with the same entropy) remains in our candidate pool for consideration.

As a result of this theorem, if $(a, b]$ 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 next 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 Sketch: Using the substitution $x = \sum_{c \in C} \gamma_{c,j}(a, z)$ and $y = \sum_{c \in C} \gamma_{c,j}(z, b)$ (see Definition 6 below for the $\gamma_{c,j}$ function), the entropy $H(z, A_j)$ can be rewritten in terms of x and y. It can be shown that the Hessian matrix $\bigtriangledown^2 H$ is negative semi-definite. Therefore, $H(x, y)$ is a concave function and hence it attains its minimum value at the corners of the domain of (x, y), which is a convex polytope. It turns out that these corners correspond to $z = a$ or $z = b$. ∎

The implication of this theorem is that 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 $c \in C$, an attribute A_j, and a set of tuples S, we define the *tuple density function* $g_{c,j}$ as:

$$g_{c,j} = \sum_{t_h \in S : c_h = c} w_h f_{h,j}$$

where w_h is the weight of the fractional tuple $t_h \in S$ (see Section III-B).

This is a weighted sum of the pdf's $f_{h,j}$ of those tuples $t_h \in S$ 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 A_j, the tuple count for class $c \in C$ in an interval $(a, b]$ is:

$$\gamma_{c,j}(a, b) = \int_a^b g_{c,j}(x)\,dx$$

The intention is that $\gamma_{c,j}(a, b)$ gives the total number of tuples within the interval $(a, b]$ 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 the tuple count for each class increases linearly in a heterogeneous interval $(a, b]$ (i.e., $\forall c \in C, \forall t \in [0, 1], \gamma_{c,j}(a, (1 - t)a + tb) = \beta_c t$ for some constant β_c). If an optimal split point falls in $(a, b]$, then an end-point of the interval is also an optimal split point.

Proof Sketch: We use the substitution $z = (1 - t)a + tb$ to rewrite the entropy $H(z, A_j)$ as a function of t. It can be

shown that $\frac{d^2H}{dt^2} \leq 0$ and hence $H(t)$ is a concave function. Consequently, H attains its minimum value at one of the extreme points $t = 0$ and $t = 1$, which correspond to $z = a$ and $z = b$, respectively. ■

One typical situation in which the condition holds is when all the pdf's follow the uniform distribution. In that case, the fraction of each tuple increases linearly in $(a, b]$, therefore the sum of a subset of them must also increase linearly. This has an important consequence: If all the pdf's are uniform distributions, then the optimal split point can be found among the $2|S|$ end-points of the intervals of the tuples in S. Theorem 3 thus reduces the number of split points to consider to $O(|S|)$.

In case the pdf's 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 $H(q, A_j)$ for all end-points $q \in Q_j$. Let $H_j^* = \min_{q \in Q_j}\{H(q, A_j)\}$ be the smallest of such end-point entropy values. 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. So, the key to this pruning technique is to find a lower bound of $H(z, A_j)$ that is not costly to compute, and yet is reasonably tight for the pruning to be effective. We have derived such a bound L_j given below. First, we introduce a few symbols to make the expression of the bound more compact and manageable: $n_c = \gamma_{c,j}(-\infty, a)$; $m_c = \gamma_{c,j}(b, +\infty)$; $k_c = \gamma_{c,j}(a, b)$; $n = \sum_{c \in C} n_c$; $m = \sum_{c \in C} m_c$; $N = n + \left(\sum_{c \in C} k_c\right) + m$; $\nu_c = \frac{n_c + k_c}{n + k_c}$ and $\mu_c = \frac{m_c + k_c}{m + k_c}$. Note that all these quantities are independent of the split point z. Our lower bound for $H(z, A_j)$ is given by:

$$L_j = -\frac{1}{N} \sum_{c \in C} [n_c \log_2 \nu_c + m_c \log_2 \mu_c + k_c \log_2(\max\{\nu_c, \mu_c\})] \quad (3)$$

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 H_j^* for each attribute A_j to prune intervals in $\mathrm{dom}(A_j)$. A better alternative is to compute a global threshold $H^* = \min_{1 \leq j \leq k} H_j^*$ 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 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 note that the entropy $H(q, A_j)$ of an end-point q is computed for two reasons. Firstly, for empty and homogeneous intervals, their end-points are the only candidates for the optimal split point. Secondly, 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%) 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 I_1, I_2, I_3, 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 I_1, I_2 and I_3.

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 Figure 5. (In this example, we ignore Theorems 1 and 2, concentrating on how end-point sampling works.) The figure shows 9 rows, illustrating 9 steps of the pruning process. Each row shows an arrowed line representing the real number line. On this line are end points (represented by crosses) or intervals (represented by line segments) drawn. Row 1 shows the intervals obtained from the domains of the pdf's. The collection of end-points of these intervals constitute the set Q_j (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 Q_j (row 2), we take a sample Q_j' (row 4) of these points. The choice of the sample size is a tradeoff between fewer entropy calculations for the end-points and a stronger pruning power. Our experiments have shown that 10% 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 Q_j' instead of Q_j. The algorithm thus operates on the intervals derived from Q_j' (row 5) instead of

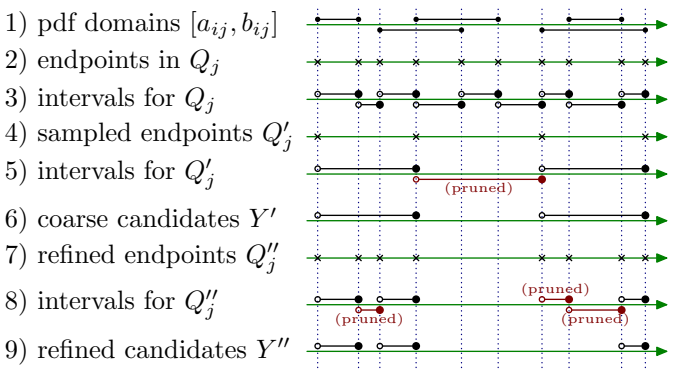


Fig. 5. Illustration of end-point sampling

those derived from Q_j (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' 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 $(q_y, q_{y+1}]$ in row 6, we bring back the original set of end-points inside the interval (row 7) and their original finer intervals (row 8). We re-invoke global-pruning again using the end-points in Q''_j (carefully caching the already calculated values of $H(q, A_j)$ for $q \in Q'_j$). The candidate set of intervals obtained after pruning is Y'' (row 9), which is a much smaller candidate than the set of candidate intervals when no end-point sampling is used. For the candidate intervals in Y'', we compute the values $H(z, A_j)$ 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, end-point sampling pays off and improves the performance of the global-pruning algorithm.

VI. Experiments on Efficiency

The algorithms described above have been implemented[8] 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 and 2GB of main memory, running Linux kernel 2.6.22 i686. Experiments on the accuracy of our novel distribution-based UDT algorithm has been presented already in Section IV-B. In this section, we focus on the pruning effectiveness of our pruning algorithms and their run-time performance.

The data sets used are the same as those used in Section IV-B. The same method is used to synthesise data uncertainty. Only Gaussian distribution is used for the experiments below. We use the parameters $s = 100$ (no. of sample points per pdf) and $w = 10\%$ (width of the pdf's domain, as a percentage of the width of the attribute's domain) as the baseline settings.

[8]The source code is available for download from http://www.cs.hku.hk/~dbgroup/UDT/.

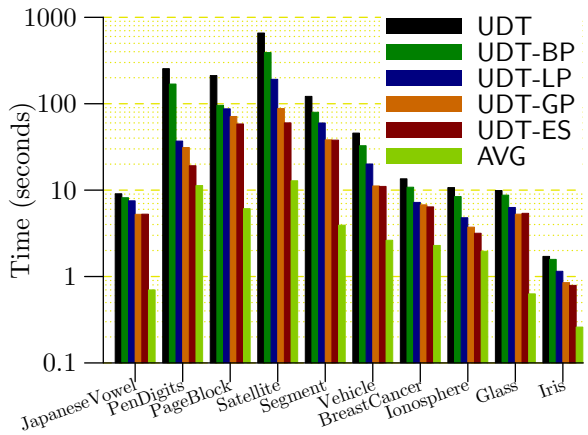


Fig. 6. Execution time

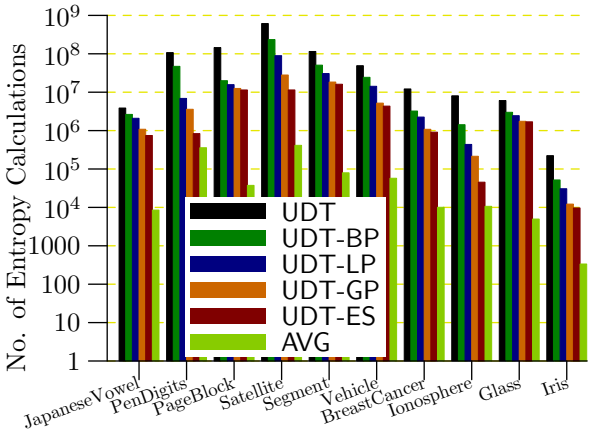


Fig. 7. Pruning effectiveness

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 Figures 8 and 9. The bars for "JapaneseVowel" in Figures 6 and 7 are given for reference only.

A. Execution Time

We first examine the execution time of the algorithms, which is charted in Figure 6. In this figure, 6 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 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. This agrees with the successive enhancements of these pruning techniques discussed in Section V. 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 IV-C). Among the distribution-based algorithms, UDT-ES is the most efficient. It takes 62% ("Ionosphere") to 865% ("Segment") more time to finish than AVG. We remark that in the experiment, each pdf is represented by 100 sample points (i.e., $s = 100$). 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 datasets "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 III). It worths to spend the extra time with the distribution-based algorithms for the higher accuracy of the resulting decision trees.

B. Pruning Effectiveness

Next, we study the pruning effectiveness of the algorithms. Figure 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 Figure 7 is also in log scale. The figure shows that our pruning techniques introduced in Section V 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 V-B). Indeed, UDT-BP only needs to perform 14%–68% of the entropy calculations done by UDT. This corresponds to a pruning of 32%–86% of the calculations. UDT-LP does even fewer calculations: only 5.4%–54% of those of UDT. By using a global pruning threshold, UDT-GP only needs to compute 2.7%–29% of entropy values compared with UDT. By pruning end-points, UDT-ES further reduces the number of entropy calculations to 0.56%–28%. 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.

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 Figure 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 increases proportionately.

D. Effects of w

Another set of experiments were 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 pdf's that are synthesised. In particular, the standard deviation chosen is a quarter of the width of the pdf's domain. Figure 9 shows the execution times of the UDT-ES algorithm. The effects of w is different for different data

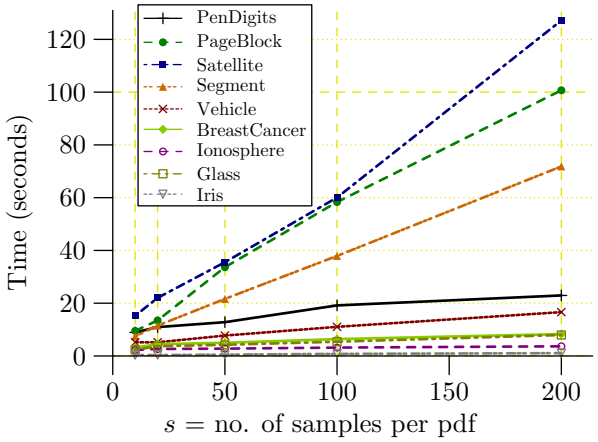


Fig. 8. Effects of s on UDT-ES

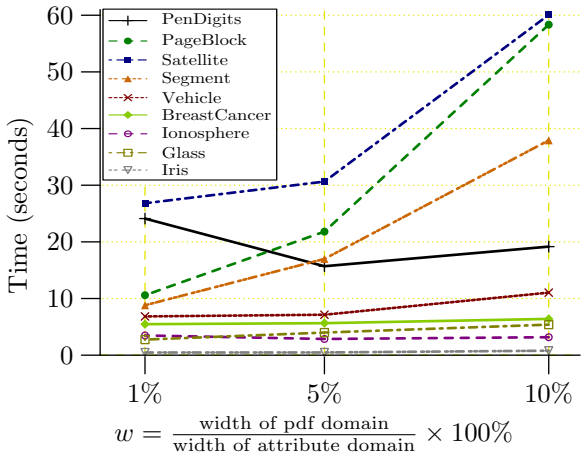


Fig. 9. Effects of w on UDT-ES

sets. In general, a larger w causes the pdf's 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 depend 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 synthesised pdf's 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 endeavour. 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 modelling 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 IV-C with the "JapaneseVowel" dataset). In the case of random noise, for example, one could fit a Gaussian distribution[5] using the sample mean and variance, thanks to the Central Limit Theorem[35].

During the search for datasets appropriate for our experiments, we have hit a big obstacle: There are few datasets with complete uncertainty information. Although many datasets with numerical attributes have been collected via repeated measurements, very often the raw data has 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" dataset (see Table II) from the UCI repository[34]. This dataset actually contains 10 uncertain numerical features collected over an *unspecified* number of repeated measurements. However, when the dataset is deposited into the repository, each of these 10 features is replaced by 3 attribute values, giving the mean, the standard score and the mean of the three largest measured values. With these 3 aggregate values, we are unable to recover the distribution of each feature. Even modelling a Gaussian distribution is impossible: These 3 aggregate values are insufficient for us to estimate the variance. Had the people preparing this dataset provided the raw measured values, we would be able to model the pdf's from these values directly, instead of injecting synthetic uncertainty and repeating this for different parameter values for w (see Section IV-D).

Now that we have established in this work that using uncertainty information modelled by pdf's 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`", "`.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 colour of a traffic light signal, which is green at the time of recording, could have changed to yellow or even red in 5 seconds, with probabilities following the programmed pattern of the signal. This is an example of uncertainty arising from data staleness. Colours of flowers recorded in a survey may divide human-visible colours into a number of categories, which may overlap with one another. Such ambiguities could be recorded as a distribution, e.g. 80% yellow and 20% 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 t_i with uncertain categorical attribute A_j, the value uncertainty can be modelled by a discrete probability distribution function $f_{i,j} : \text{dom}(A_j) \rightarrow [0, 1]$ satisfying $\sum_{x \in \text{dom}(A_j)} f_{i,j}(x) = 1$. This is analogous to the case of uncertain numerical attribute. An internal node n in the decision tree corresponding to a categorical attribute A_j is not associated with a split point, though. Rather, n has many child nodes, each corresponding to a distinct value in $\text{dom}(A_j)$.[9] The test to perform at node n is to check the value of A_j 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 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.

To evaluate the entropy of a categorical attribute A_j, we (fractionally) split the tuples in question into a set of buckets $\{B_v | v \in \text{dom}(A_j)\}$. Tuple t_x is copied into B_v as a new tuple t_y with weight $w_y = f_{x,j}(v)$ if and only if $w_y > 0$. The pdf's of t_y are inherited from t_x, except for attribute A_j, which is set to $f_{y,j}(v) = 1$ and $f_{y,j}(w) = 0$ for all $w \neq v$. The entropy for the split on A_j 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 pdf's

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

[9] Typically, $\text{dom}(A_j)$ has a relatively small cardinality.

However, in case the pdf's are unbounded, the pruning techniques can as well be applied to some artificial "end-points". For example, suppose we are handling attribute A_j. For each class c, we could treat the tuple count $\gamma_{c,j}(-\infty, t)$ 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 $9|C|$ 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. Generalising the Theorems

In Section V-A, 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 space limitations. Consequently, the pruning techniques (Section V) can been applied to Gini index as well. A different lower bound formula for L_j is needed, though:

$$\begin{aligned} L_j^{(\mathrm{Gini})} &= 1 - \tfrac{1}{N}\Big(n\textstyle\sum_{c\in C}\nu_c^2 + m\sum_{c\in C}\mu_c^2 \\ &\quad + \min\Big\{\big[\textstyle\sum_{c\in C} k_c(\nu_c^2+\mu_c^2)\big], \\ &\quad\quad k\cdot\max\Big\{\big[\textstyle\sum_{c\in C}\nu_c^2\big], \big[\textstyle\sum_{c\in C}\mu_c^2\big]\Big\}\Big\}\Big) \end{aligned} \tag{4}$$

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 V-B) and end-point sampling (Section V-C) 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 extended the model of decision-tree classification to accommodate data tuples having numerical attributes with uncertainty described by arbitrary pdf's. We have modified classical decision tree building algorithms (based on the framework of C4.5[3]) to build decision trees for classifying such data. We have found empirically that when suitable pdf's 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. 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 generalisations 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.

References

[1] R. Agrawal, T. Imielinski, and A. N. Swami, "Database mining: A performance perspective," *IEEE Trans. Knowl. Data Eng.*, vol. 5, no. 6, pp. 914–925, 1993.

[2] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[3] ——, *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993, ISBN 1-55860-238-0.

[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, pp. 189–202, 2000.

[5] G. L. Freed and J. K. Fraley, "25% "error rate" in ear temperature sensing device," *Pediatrics*, vol. 87, no. 3, pp. 414–415, Mar. 1991.

[6] O. Wolfson and H. Yin, "Accuracy and resource consumption in tracking and location prediction," in *SSTD*, ser. Lecture Notes in Computer Science, vol. 2750. Santorini Island, Greece: Springer, 24-27 Jul. 2003, pp. 325–343.

[7] W. Street, W. Wolberg, and O. Mangasarian, "Nuclear feature extraction for breast tumor diagnosis," in *SPIE*, vol. 1905, San Jose, CA, U.S.A., 1993, pp. 861–870. [Online]. Available: http://citeseer.ist.psu.edu/street93nuclear.html

[8] N. N. Dalvi and D. Suciu, "Efficient query evaluation on probabilistic databases," *The VLDB Journal*, vol. 16, no. 4, pp. 523–544, 2007.

[9] E. Hung, L. Getoor, and V. S. Subrahmanian, "Probabilistic interval XML," *ACM Transactions on Computational Logic (TOCL)*, vol. 8, no. 4, 2007.

[10] A. Nierman and H. V. Jagadish, "ProTDB: Probabilistic data in XML," in *VLDB*. Hong Kong, China: Morgan Kaufmann, 20-23 Aug. 2002, pp. 646–657.

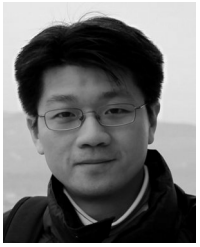
[11] J. Chen and R. Cheng, "Efficient evaluation of imprecise location-dependent queries," in *ICDE*. Istanbul, Turkey: IEEE, 15-20 Apr. 2007, pp. 586–595.

[12] M. Chau, R. Cheng, B. Kao, and J. Ng, "Uncertain data mining: An example in clustering location data," in *PAKDD*, ser. Lecture Notes in Computer Science, vol. 3918. Singapore: Springer, 9–12 Apr. 2006, pp. 199–204.

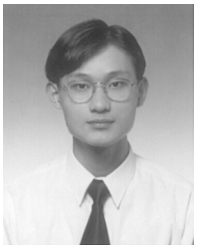
[13] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. S. Vitter, "Efficient indexing methods for probabilistic threshold queries over uncertain data," in *VLDB*. Toronto, Canada: Morgan Kaufmann, 31 Aug.–3 Sept. 2004, pp. 876–887.

[14] R. Cheng, D. V. Kalashnikov, and S. Prabhakar, "Querying imprecise data in moving object environments," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 9, pp. 1112–1127, 2004.
[15] W. K. Ngai, B. Kao, C. K. Chui, R. Cheng, M. Chau, and K. Y. Yip, "Efficient clustering of uncertain data," in *ICDM*. Hong Kong, China: IEEE Computer Society, 18–22 Dec. 2006, pp. 436–445.
[16] S. D. Lee, B. Kao, and R. Cheng, "Reducing UK-means to K-means," in *The 1st Workshop on Data Mining of Uncertain Data (DUNE), in conjunction with the 7th IEEE International Conference on Data Mining (ICDM)*, Omaha, NE, USA, 28 Oct. 2007.
[17] H.-P. Kriegel and M. Pfeifle, "Density-based clustering of uncertain data," in *KDD*. Chicago, Illinois, USA: ACM, 21–24 Aug. 2005, pp. 672–677.
[18] C. K. Chui, B. Kao, and E. Hung, "Mining frequent itemsets from uncertain data," in *PAKDD*, ser. Lecture Notes in Computer Science, vol. 4426. Nanjing, China: Springer, 22-25 May 2007, pp. 47–58.
[19] C. C. Aggarwal, "On density based transforms for uncertain data mining," in *ICDE*. Istanbul, Turkey: IEEE, 15-20 Apr. 2007, pp. 866–875.
[20] O. O. Lobo and M. Numao, "Ordered estimation of missing values," in *PAKDD*, ser. Lecture Notes in Computer Science, vol. 1574. Beijing, China: Springer, 26-28 Apr. 1999, pp. 499–503.
[21] L. Hawarah, A. Simonet, and M. Simonet, "A probabilistic approach to classify incomplete objects using decision trees," in *DEXA*, ser. Lecture Notes in Computer Science, vol. 3180. Zaragoza, Spain: Springer, 30 Aug.-3 Sep. 2004, pp. 549–558.
[22] J. R. Quinlan, "Learning logical definitions from relations," *Machine Learning*, vol. 5, pp. 239–266, 1990.
[23] Y. Yuan and M. J. Shaw, "Induction of fuzzy decision trees," *Fuzzy Sets Syst.*, vol. 69, no. 2, pp. 125–139, 1995.
[24] M. Umanol, H. Okamoto, I. Hatono, H. Tamura, F. Kawachi, S. Umedzu, and J. Kinoshita, "Fuzzy decision trees by fuzzy ID3 algorithm and its application to diagnosis systems," in *Fuzzy Systems, 1994*, vol. 3. IEEE World Congress on Computational Intelligence, 26-29 Jun. 1994, pp. 2113–2118.
[25] C. Z. Janikow, "Fuzzy decision trees: issues and methods," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 28, no. 1, pp. 1–14, 1998.
[26] C. Olaru and L. Wehenkel, "A complete fuzzy decision tree technique," *Fuzzy Sets and Systems*, vol. 138, no. 2, pp. 221–254, 2003.
[27] T. Elomaa and J. Rousu, "General and efficient multisplitting of numerical attributes," *Machine Learning*, vol. 36, no. 3, pp. 201–244, 1999.
[28] U. M. Fayyad and K. B. Irani, "On the handling of continuous-valued attributes in decision tree generation," *Machine Learning*, vol. 8, pp. 87–102, 1992.
[29] 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.
[30] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Wadsworth, 1984.
[31] T. Elomaa and J. Rousu, "Necessary and sufficient pre-processing in numerical range discretization," *Knowledge and Information Systems*, vol. 5, no. 2, pp. 162–182, 2003.
[32] L. Breiman, "Technical note: Some properties of splitting criteria," *Machine Learning*, vol. 24, no. 1, pp. 41–47, 1996.
[33] T. M. Mitchell, *Machine Learning*. McGraw-Hill, 1997, ISBN 0070428077.
[34] A. Asuncion and D. Newman, "UCI machine learning repository," 2007. [Online]. Available: http://www.ics.uci.edu/~mlearn/MLRepository.html
[35] R. E. Walpole and R. H. Myers, *Probability and Statistics for Engineers and Scientists*. Macmillan Publishing Company, 1993.
[36] S. Tsang, B. Kao, K. Y. Yip, W.-S. Ho, and S. D. Lee, "Decision trees for uncertain data," in *ICDE*, Shanghai, China, 29 Mar.–4 Apr. 2009, pp. 441–444.
[37] *Proceedings of the 23nd International Conference on Data Engineering*. Istanbul, Turkey: IEEE, 15-20 Apr. 2007.

Smith Tsang works in MileSCAN Technologies Limited. He received his Bachelor of Engineering and Master of Philosophy degrees from The University of Hong Kong in 2006 and 2009 respectively. His research interests include data mining and uncertain data classification.



Ben Kao received the B.Sc. degree in computer science from the University of Hong Kong in 1989, the Ph.D. degree in computer science from Princeton University in 1995. From 1989–1991, he was a teaching and research assistant at Princeton University. From 1992–1995, he was a research fellow at Stanford University. He is currently associate professor of the Department of Computer Science at the University of Hong Kong. His research interests include database management systems, data mining, real-time systems, and information retrieval systems.



Kevin Y. Yip is a post-doctoral associate at Yale University. He received his B.Engg in Computer Engineering and M.Phil in Computer Science from the University of Hong Kong, in 1999 and 2004, respectively. He received his PhD in Computer Science from Yale University in 2009. His current research interest is in bioinformatics, with a special focus on biological network inference and analysis using data mining and machine learning techniques.

Wai-Shing Ho teaches in the Department of Computer Science, The University of Hong Kong. He received his Bachelor of Engineering and Doctor of Philosophy degrees from The University of Hong Kong in 1999 and 2005 respectively. His research interests include uncertain data classification and clustering, and online analytical processing of sequence data.

Sau Dan Lee is a Post-doctoral Fellow at the University of Hong Kong. He received his Ph.D. degree from the University of Freiburg, Germany in 2006 and his M.Phil. and B.Sc. degrees from the University of Hong Kong in 1998 and 1995. He is interested in the research areas of data mining, machine learning, uncertain data management and information management on the WWW. He has also designed and developed backend software systems for e-Business and investment banking.