# APPENDIX A

# HANDLING PARTIALLY DEFINED ATTRIBUTES

In this appendix we describe the method we adopted for handling examples that have undefined values for some attributes. The method described in this appendix applies to all algorithms described in this dissertation: ID3, GID3, GID3*, and O-BTREE. The method targets handling a specific type of undefined values: partially defined attributes. Thus it is intended for use in domains were some attributes were not defined at an earlier stage, some training data was collected, then new attributes were added. However, the method can be used to handle missing attribute values for any other reason: noise, data corruption, incomplete data acquisition, etc.

## A.1 Introduction

In an industrial setting, process data gets collected over time. The engineer first decides what data to collect (what attributes to measure) and then begins recording the results of experiments. However, at some later time, the engineer may realize that additional attributes should be recorded as well. Typically, increased familiarity with a set of experiments results in the discovery of new (possibly more relevant/important) attributes. This results in a *nonhomogeneous* collection of data points in which the earlier data points have undefined values for the attributes that were collected later.

One approach to deal with such data is to discard older data points and keep only the completely defined data. However, this is wasteful since it results in the total loss of information contained in earlier experiments. Furthermore, it may be the case that experiments are expensive or time consuming making it undesirable to repeat the older experiments to get the values of the new attribute(s). Another approach would be to use the entire data but ignore the new attributes. However, this would be counterproductive since it ignores the information contained in the new attributes: the engineer must have had good reason to start recording the values of the new attributes.

Having ruled out the two simple approaches we are left with only one option: modify the learning algorithm so that it can deal with partially defined attributes. We would like the learning algorithm to use the new attribute whenever that is necessary or appropriate. Furthermore, as more "new" data is collected, the algorithm should start ignoring older data (again when necessary and appropriate). Below we discuss different possible approaches to this problem and the approach we decided to adopt.

## A.2 Possible Approaches

There are three issues that need consideration when changing a tree generation algorithm to make it capable of handling partially defined attributes. We need to decide what is to be done when

1. evaluating the attribute for selection for branching at a node,

2. partitioning the data along the branches if a partially defined attribute is selected, and

3. classifying a test example at a node that tests on the value of a partially defined attribute.

The problem we are addressing is a special instance of the general problem of partially defined attributes. It differs in that future examples that are to be classified using the generated decision tree will have all their attribute values defined. This is due to the fact that once an attribute is defined, all examples collected in the future are assumed to have a value for that attribute. Consequently, we do not have to address issue 3 above. This further specifies and simplifies the problem at hand. In our case we only anticipate encountering undefined attribute values in the training set. If, when using the tree to classify test examples, we attempt to test the value of an attribute that is not defined, we simply fail to classify it. Other approaches for dealing with such a situation more effectively are described in [95].

### A.2.1 Attribute Evaluation

We first list possible ways of dealing with a partially defined attribute during the attribute evaluation stage of the algorithm. We discuss the problems with each of the methods in Section A.2.3 below.

**A.** Ignore examples with undefined values when evaluating the undefined attribute. The merit of an attribute is then evaluated based only on the subset of examples for which the attribute has a value.

**B.** Fill in the values of the attribute for the examples that have that value missing. Essentially the algorithm tries to "guess" what the missing values should be. In this case, statistical methods could be employed to estimate the missing values. Alternatively, one could create a learning subproblem to determine the missing values. In this case, the set of examples whose values are defined are used as a training set to generate a tree for predicting the values of the partially defined attribute. The tree is then used to predict the missing values on the remaining subset of examples.

**C.** Perform A above but then reduce the gain measure assigned to the attribute to reflect the fact that the gain was based on a subset of the data and not the entire set. The attribute is thus "penalized" because it ignores some of the information available in the data.

**D.** Create a separate branch for undefined values of an attribute.

**E.** Consider a partially defined attribute at a node only if all the examples at the node have defined values for the attribute. This excludes the partially defined attribute from consideration at the root node, and may exclude it from consideration at all nodes in the tree (depending on the data).

### A.2.2 Partitioning Data when Branching

Once we decide on a method for evaluating the attributes (including the partially defined attributes) at a node, we have to decide how the data is to be partitioned along the branches created for the various values of the selected attribute if it is partially defined on the data. We list possible methods to achieve this:

**A2.** Discard the examples that have undefined values for the chosen attribute (the analogue for A above).

**B2.** "Guess" what the value of each example should be and move each example along the matching branch (the analogue of B above).

**D2.** Move all undefined examples along an undefined branch (this is only applicable in case D above).

**F2.** Move each example whose value is undefined for the selected attribute along each branch out of the node.

### A.2.3 Discussion of Various Methods

Simply ignoring the examples with undefined values, as suggested in A, is not valid because this will bias the algorithm towards the new attributes while disregarding the information available in the older data. Method B (and similarly B2) is both unpredictable and may introduce errors in the data. An incorrect guess for a value of an attribute may result in changing the resulting tree dramatically. Method D (and similarly D2) is not suitable in our case because undefined examples will never be encountered when the tree is used to classify future examples. Thus branches for undefined values will simply never be used. Method E is reasonable but essentially prevents the algorithm from using new attributes at the higher levels of the tree. Furthermore, there may be cases when all nodes in the tree will get a few undefined examples. In this case method E will cause the algorithm to completely ignore the new attributes. Finally, method F2 attempts to maximize the usage of all available information, however it overdoes this. By moving undefined examples along all branches, the tree is likely to grow much larger and the likelihood of making erroneous classification increases.

It is preferable to use a conservative strategy (one that ignores some information but does not actively introduce errors) rather than using a risky method that uses more information. We therefore decided to adopt a modification of the C-A2 method. We justify this choice in the next section.

## A.3 The Adopted Approach

After considering the different methods, we decided to use method C for attribute evaluation and method A2 for partitioning the examples along the branches. Method C prescribes that the attribute be evaluated based only on the subset of examples that have defined values for the attribute. The merit assigned to the attribute is then reduced to reflect the fact that some examples were ignored during the evaluation. We decided to reduce the merit by the percentage of examples that were ignored (i.e. the proportion of examples that have undefined values for the attribute). This method intuitively achieves the desired goals:

1. It discourages the algorithm from using the partially defined attributes when the proportion of undefined examples is high (i.e. when the proportion of examples for which the new attribute was recorded is low). The partially defined attributes will thus tend to appear only towards the lower portion of the tree.

2. As the number of new examples grows, the bias against using the partially defined attributes at the higher levels of the tree diminishes. Eventually, the

algorithm will essentially exhibit no bias towards avoiding the "new" attributes at the root.

3. The information included in the old examples is not ignored since they serve to penalize the new attribute. At the same time, the other (old) attributes are evaluated based on the information available in both the old and new data.

Method A2 for handling the examples that are undefined for the selected attribute was chosen because:

1. Ignoring the undefined examples is the most conservative strategy. It does ignore information but that price is paid only when the new attributes are favoured by the selection (merit) measure to the extent that they overcome the penalty incurred on them.

2. As more "new" examples are collected, and the algorithm begins to select the partially defined attributes towards the higher levels of the tree, the algorithm essentially begins to ignore older examples. Eventually, old examples will naturally get completely ignored. This is a desirable feature since we want the learning algorithm to "focus" more on the completely defined (newer) examples as their number grows much larger than the (partially defined) older examples.

### A.3.1 Requirements of the Method

An undefined attribute value must appear as a '?' for either discrete or continuous-valued attributes. The users have therefore to make sure that '?' is not an actual value in the range of a nominal attribute.

As defined, the method assumes that the merit assigned to each attribute in preparation for attribute selection is non-negative and that a higher value implies better merit.

Assume that the algorithm is currently evaluating the attributes at a node containing a set $S$ of examples. Assume that attribute $A$ is being evaluated and that some of the examples in $S$ have undefined values for $A$. The algorithm first takes the subset $S'$ of $S$ consisting of all examples that have defined values. The proportion of defined examples at the node, for attribute $A$ is

$$r = \frac{|S'|}{|S|} \leq 1.0$$

$r$ will be the penalty by which the merit of $A$ will be reduced later. In case of ID3, GID3, GID3*, and ID3-BIN the merit is the information gain. In case of ID3-IV, it is the gain ratio and in the case of O-BTREE it is the ORT measure. Since we have removed some examples of $S$ to get $S'$, we check to make sure that the set $S'$ contains examples of more than one class. If all the examples in $S'$ are of one class then attribute $A$ is of no use to us and we should exclude it from consideration for branching.

Once a partially defined attribute is chosen for branching at a node, the examples at the node having undefined values for that attribute are discarded and will never appear in the subtree under the node. This is the only point where the algorithm discards information. However, this is the only conservative option at this point.

For a more detailed discussion of this method and for an example on partially defined data obtained from Hughes, the reader is referred to [26]. The description in this appendix is intended to serve as documentation for the experiments conducted in this dissertation.

# APPENDIX B

# ENTROPY AND REFINED PARTITIONS

In this appendix we prove that if attribute $A$ induces a partition $\pi_1$ on a data set $S$,

$$\pi_1 = \{S_1, S_2, \ldots, S_r\}$$

and attribute $A'$ induces the partition $\pi_2$ on the same data set S:

$$\pi_2 = \{S'_1, S'_2, \ldots, S'_t\}$$

then if $\pi_2$ is a refinement on $\pi_1$, the information gain of $A'$ will be the same or higher than $Gain(A, S)$.

We shall assume that $S$ consists of $N = |S|$ examples of $k$ classes. Recall by the definition of $Gain$ in Equation 4.2

$$Gain(A, S) = \mathrm{Ent}(S) - E(A, S).$$

Since the class information entropy of $S$, $\mathrm{Ent}(S)$ is a constant for fixed $S$, all we have to establish is the following lemma:

**Lemma B.0.1** *If the partition induced on $S$ by attribute $A'$ is a refinement of the partition induced on $S$ by attribute $A$, then*

$$Ent(A', S) \leq Ent(A, S).$$

**Proof:** Let $\pi_1$ be

$$\pi_1 = \{S_1, S_2, \ldots, S_r\}.$$

Since $\pi_2$ is a refinement of $\pi_1$, we can rewrite $\pi_2$ as

$$\pi_2 = \{S_{11}, \ldots, S_{1,l_1}, S_{21}, \ldots, S_{2l_2}, S_{r1}, \ldots, S_{rl_r}\}$$

for some appropriate $l_i \geq 1$, $1 \leq i \leq r$, where for each $i$,

$$\bigcup_{j=1}^{l_i} S_{ij} = S_i$$

Thus the $S_{ij}$ partition the subset $S_i$. Now,

$$\begin{aligned} E(A, S) &= \sum_{i=1}^{r} \frac{|S_i|}{N} \mathrm{Ent}(S_i) \\ E(A', S) &= \sum_{i=1}^{r} \sum_{j=1}^{l_i} \frac{|S_{ij}|}{N} \mathrm{Ent}(S_{ij}) \end{aligned}$$

To prove the lemma, all we need to show is that for each $i$, $1 \leq i \leq r$,

$$\sum_{j=1}^{l_i} |S_{ij}| \mathrm{Ent}(S_{ij}) \quad \leq \quad |S_i| \mathrm{Ent}(S_i).$$

Equivalently, we need to show that for each $i$,

$$\mathrm{Ent}(S_i) \geq \sum_{j=1}^{l_i} \frac{|S_{ij}|}{|S_i|} \mathrm{Ent}(S_{ij}).$$

However, this follows immediately by Lemma B.0.2. $\square$

To proceed further, we need to introduce some notation. Given a data set $S$ of training examples from $k$ classes and an attribute $B$ with values $\{b_1, b_2, \ldots, b_r\}$. The attribute $B$ induces the partition $\{S_1, S_2, \ldots, S_r\}$ on $S$, where each $S_i \subseteq S$ consists of examples in $S$ that have value $B_i$ for $B$. We shall use $\mathrm{P}(C_i)$, $i = 1, \ldots, k$ to denote the proportion of examples in $S$ that have class $C_i$. This may also be thought of as the "probability" of coming across class $C_i$ in $S$. Similarly, we use $\mathrm{P}(b_j)$ to denote the proportion of examples in $S$ that have value $b_j$ for $B$, i.e.,

$$\mathrm{P}(b_j) = \frac{|S_j|}{N}.$$

We use $\mathrm{P}(C_i|b_j)$ to denote the proportion of examples in $S_j$ that have class $C_i$. This can be thought of as the probability of coming across class $C_i$ given that $B = b_j$ (the conditional probability of $C_i$ given $B = b_j$). Finally, we define $\mathrm{P}(C_i, b_j)$ to be the proportion of examples in $S$ that have class $C_i$ and value $b_j$ for attribute $B$. Thus, it may be thought of as the joint probability of the class being $C_i$ and the value of $B$ being $b_j$ for an example in $S$. We denote the number of examples in $S$ that have class $C_i$ and $B$-value $b_j$ by $c_{ij}$. Hence,

$$\mathrm{P}(C_i, b_j) = \frac{c_{ij}}{|S|} = \frac{c_{ij}}{N}.$$

As a consequence of these definitions, we have

$$\begin{aligned}\mathrm{P}(C_i|b_j) &= \frac{c_{ij}}{|S_j|} \\ &= \frac{c_{ij}/N}{|S_j|/N} \\ &= \frac{\mathrm{P}(C_i, b_j)}{\mathrm{P}(b_j)}\end{aligned}$$

which means that we can use the definition of Bayes' rule without problems. Using this notation, we can rewrite the definition of $E(B, S)$ as follows:

$$\begin{aligned}E(B,S) &= \sum_{j=1}^{r} \frac{|S_j|}{N}\mathrm{Ent}(S_j) \\ &= \sum_{j=1}^{r} \mathrm{P}(b_j)\mathrm{Ent}(S_j) \\ &= \sum_{j=1}^{r} \mathrm{P}(b_j)\left[\sum_{i=1}^{k} -\mathrm{P}(C_i|b_j)\log_2\left(\mathrm{P}(C_i|b_j)\right)\right]\end{aligned}$$

**Fact 2** *For any real number $z > 0$, $\ln(z) \leq (z-1)$, where $\ln(z)$ denotes the natural logarithm of $z$, i.e.,*

$$\ln(z) = \log_e(z) = \frac{\log_2(z)}{\log_2(e)}.$$

**Lemma B.0.2** *For any training set $S$ and attribute $B$ with values $\{b_1, \ldots, b_r\}$ the information gain of $B$ with respect to $S$, $Gain(B, S)$ is non-negative:*

$$Gain(B,S) = Ent(S) - E(B,S) \;\; \geq \;\; 0.$$

**Proof:** Using the above notation, we need to show that

$$\begin{aligned}
Gain(B,S) &= \mathrm{Ent}(S) - E(B,S) \\
&= -\sum_{i=1}^{k} \mathrm{P}(C_i)\log_2(\mathrm{P}(C_i)) - \sum_{j=1}^{r} \mathrm{P}(b_j)\mathrm{Ent}(S_j) \\
&= \sum_{i=1}^{k} \mathrm{P}(C_i)\log_2\left(\frac{1}{\mathrm{P}(C_i)}\right) + \sum_{j=1}^{r} \mathrm{P}(b_j)\left[\sum_{i=1}^{k} \mathrm{P}(C_i|b_j)\log_2(\mathrm{P}(C_i|b_j))\right] \\
&\geq 0.
\end{aligned}$$

Further manipulation of the latter expression for *Gain*, applying Bayes' rule, and merging the summations we get

$$\begin{aligned}
Gain(B,S) &= \sum_{i=1}^{k} \mathrm{P}(C_i)\log_2\left(\frac{1}{\mathrm{P}(C_i)}\right) + \sum_{j=1}^{r} \mathrm{P}(b_j)\left[\sum_{i=1}^{k} \frac{\mathrm{P}(C_i,b_j)}{\mathrm{P}(b_j)}\log_2(\mathrm{P}(C_i|b_j))\right] \\
&= \sum_{i=1}^{k} \mathrm{P}(C_i)\log_2\left(\frac{1}{\mathrm{P}(C_i)}\right) + \sum_{j=1}^{r}\sum_{i=1}^{k} \mathrm{P}(C_i,b_j)\log_2(\mathrm{P}(C_i|b_j)) \\
&= \sum_{j=1}^{r}\sum_{i=1}^{k}\left[\mathrm{P}(C_i,b_j)\log_2\left(\frac{1}{\mathrm{P}(C_i)}\right) + \mathrm{P}(C_i,b_j)\log_2(\mathrm{P}(C_i|b_j))\right] \\
&= \sum_{j=1}^{r}\sum_{i=1}^{k} \mathrm{P}(C_i,b_j)\log_2\left(\frac{\mathrm{P}(C_i|b_j)}{\mathrm{P}(C_i)}\right) \\
&= \frac{1}{\log_2(e)}\sum_{j=1}^{r}\sum_{i=1}^{k} \mathrm{P}(C_i,b_j)\ln\left(\frac{\mathrm{P}(C_i|b_j)}{\mathrm{P}(C_i)}\right)
\end{aligned}$$

Let us examine the quantity $-Gain(B,S)$,

$$\begin{aligned}
-Gain(B,S) &= -\frac{1}{\log_2(e)}\sum_{j=1}^{r}\sum_{i=1}^{k} \mathrm{P}(C_i,b_j)\ln\left(\frac{\mathrm{P}(C_i|b_j)}{\mathrm{P}(C_i)}\right) \\
&= \frac{1}{\log_2(e)}\sum_{j=1}^{r}\sum_{i=1}^{k} \mathrm{P}(C_i,b_j)\ln\left(\frac{\mathrm{P}(C_i)}{\mathrm{P}(C_i|b_j)}\right)
\end{aligned}$$

Using Fact 2, and substituting $\ln\left(\frac{\mathrm{P}(C_i)}{\mathrm{P}(C_i|b_j)}\right)$ by its larger couterpart we get

$$\begin{aligned}
-Gain(B,S) &\leq \frac{1}{\log_2(e)}\sum_{j=1}^{r}\sum_{i=1}^{k} \mathrm{P}(C_i,b_j)\left(\frac{\mathrm{P}(C_i)}{\mathrm{P}(C_i|b_j)} - 1\right) \\
&= \frac{1}{\log_2(e)}\sum_{j=1}^{r}\sum_{i=1}^{k} \mathrm{P}(C_i,b_j)\left(\frac{\mathrm{P}(C_i)\mathrm{P}(b_j)}{\mathrm{P}(C_i,b_j)} - 1\right) \\
&= \frac{1}{\log_2(e)}\sum_{j=1}^{r}\sum_{i=1}^{k} \mathrm{P}(C_i)\mathrm{P}(b_j) - \mathrm{P}(C_i,b_j)
\end{aligned}$$

$$\begin{aligned} &= \frac{1}{\log_2(e)}\left[\sum_{j=1}^{r} \mathrm{P}(b_j) - \sum_{j=1}^{r} \mathrm{P}(b_j)\right] \\ &= 0. \end{aligned}$$

Which gives us the desired result of $Gain(B, S) \geq 0$. $\square$