Determining the Incremental Worth of Members of an Aggregate Set through Difference-Based Induction

Avelino J. Gonzalez
University of Central Florida
PO Box 162450
Orlando, FL 32816-2450
ajg@ece.engr.ucf.edu

Sylvia Daroszewski
Harris Corporation
Palm Bay, FL

Howard J. Hamilton
Dept. of Computer Science
University of Regina
Regina, Canada
hamilton@cs.uregina.ca

Abstract

Calculating the incremental worth or weight of the individual components of an aggregate set when only the total worth or weight of the whole set is known is a problem common to several domains. In this article we describe an algorithm capable of inducing such incremental worth from a database of similar (but not identical) aggregate sets. The algorithm focuses on finding aggregate sets in the database that exhibit minimal differences in their corresponding components (referred to here as *attributes* and their *values*). This procedure isolates the dissimilarities between nearly similar aggregate sets so that any difference in worth between the sets is attributed to these dissimilarities. In effect, this algorithm serves as a mapping function that maps the makeup and overall worth of an aggregate set into the incremental worth of its individual attributes. It could also be categorized as a way of calculating interpolation vectors for the attributes in the aggregate set.

Introduction

Knowing the relative contribution of individual components of an aggregate set in terms of their worth (or cost, weight, etc.) to the total worth (or cost, weight, etc.) of the whole set can be important in domains such as engineering analysis, scientific discovery, accounting, and real estate appraisal. A slightly different but very similar problem is that of obtaining the incremental worth of such individual components. This task can be a difficult one when only the total worth of the whole set is known.

The work described in this article focuses on an approach to calculate the *incremental* worth of components of aggregate sets. This is possible when a database contains many aggregate sets that are homogeneous (i.e., have the same attributes), but dissimilar (i.e., have different values). An algorithm is formulated which finds aggregate sets in the database that exhibit minimal differences in their attributes' values. This procedure strives to isolate the single differentiating attribute/value combination of nearly similar aggregate sets.

Any difference in worth between the sets can be attributed to this singular difference. For this reason, the algorithm is called the *difference-based induction algorithm*, or DBI algorithm for short.

Definition of Terms

An *aggregate set* represents a collection, assembly or grouping of member components which have a common nature or purpose. An aggregate set is made up of these aforementioned components. The term *attribute* will be used to signify a component of an aggregate set, and each attribute is said to have a particular *value*. A group of such attribute/value combinations can singularly describe a specific aggregate set.

The components of an aggregate set act as integral members, which individually add worth to the aggregate set. However, their individual contributions to the total worth of the aggregate set may not be explicitly known. We will use the term *worth* here generically to mean a real number representing the worth, weight, cost, price or any other similarly quantifiable factor.

An *aggregate set*, **AG**, is therefore defined as a pair consisting of a set containing its components (called *attributes*, **A**), and the total worth of the aggregate set, (**worth**). Therefore, the i^{th} aggregate set AG_i (of **n** total aggregate sets) can be represented by a pair as

$$AG_i = \langle A_i \ \ worth_i \rangle.$$

A_i is further defined as the group of attributes for the set AG_i, such that for **m** attributes in AG_i:

$$A_i = \{a_{i,1}\ a_{i,2}\ a_{i,3}\ a_{i,4}\ a_{i,5}\ \ldots\ a_{i,m}\}$$

where $a_{i,j}$ is the *name* of the j^{th} attribute of the i^{th} aggregate set.

Furthermore, each attribute, $a_{i,j}$, represents another pair depicting the *value* of the attribute ($v_{i,j}$), and its individual *worth* ($w_{i,j}$). The individual worth of an attribute is the amount that this attribute contributes to the overall worth of the aggregate set ($worth_i$). In the problems described here, $w_{i,j}$ is typically not known.

Care should be taken to not confuse the terms "value" and "worth". *Value* is the value inherent with that attribute, much like attribute-value pairs in frames or the

value of an attribute in a relational database. The value of an attribute can be either discrete or continuous. Discrete values can be binary, such as *present* or *not-present*, Yes or No, etc., or non-binary. A non-binary attribute can take on one of several possible values, which may be numerical or symbolic in nature. Non-binary discrete attributes (such as the *engine-size* of a car) typically have a relatively small set of possible values (4-cyl, 6-cyl, 8-cyl.). Continuous attributes, on the other hand, such as the time to accelerate from 0 to 60 MPH, can take on a value from a much larger domain. The range of the continuous domain values can be divided into several sub-intervals (i.e., 5-6 secs, 7-8 secs, 9-10 secs, 10-11 secs, 12-13 secs and so on) to convert a continuous value into a discrete one.

Thus, the pair defining the j^{th} attribute a_j of the aggregate set AG_i is

$$A_j = <v_{i,j} \ w_{i,j}>.$$

The value of an attribute, $v_{i,j}$ must come from a pre-defined set of possible values or range,

$$V_{i,j} \text{ (element of) } \{pv_1 \ pv_2 \ pv_3 \ pv_4 \ pv_5 \ldots pv_j\}$$

(if discrete) or

$$pv_{min} <= v_{i,j} <= pv_{max}$$

(if continuous range)

We furthermore define the subtle difference between *relative worth* and *incremental worth*. If $w_{i,j}$ were to represent the relative worth, it must be true that

$$worth_i = (w_{i,1} + w_{i,2} + w_{i,3} + \ldots + w_{i,m}).$$

However, this is not a necessary condition if $w_{i,j}$ represents the incremental worth of an attribute/value combination. Our algorithm does not require the above expression to hold.

A database, **DB**, containing **n** aggregate sets can be defined as

$$DB = \{AG_1 \ AG_2 \ AG_3 \ldots \ AG_n\}.$$

The DBI algorithm, however, assumes that all aggregate sets in **DB** have exactly the same attributes. Any difference between the aggregate sets is reflected by the values of its attributes. However, if an aggregate set were to lack a particular attribute, this can be represented by assigning a value of **false** or **none** to the missing attribute. Using the same definition formalism, this homogeneity of aggregate sets could be represented as:

$$\text{(for all) i, } a_{i,j} = a_{i+1,j}$$

This would allow the following simplification in our definition:

$$A = \{a_1 \ a_2 \ a_3 \ a_4 \ .. \ a_j \ .. \ a_m\}$$

However, since the value of an attribute can differ between the various aggregate sets, the value and its worth remain as previously defined:

$$a_j = <v_{i,j} \ w_{i,j}>.$$

Thus, it is not required that

$$\text{(for all) i, } v_{i,j} = v_{i+1,j} \text{ and } w_{i,j} = w_{i+1,j}$$

It is the goal of our work to determine the incremental worth of a specific attribute/value combination in an aggregate set. Since the domains where this information would be highly useful are not fields of exact sciences, the relative or incremental worth of an individual attribute/value combination can vary somewhat between various aggregate sets in the database. Our goal, therefore, is to find the *average* incremental or relative worth of a particular attribute/value combination based on the overall worths of several aggregate sets.

Lastly, the *critical attribute* is defined as the attribute whose incremental worth is to be discovered for different value assignments.

To put these definitions in proper perspective, we will apply them to a description of the value of cars in an automobile dealership. A dealership may have a large number of automobiles (aggregate sets) in their inventory (database). Each automobile has a set of attributes (*engine size, tires, color, doors, power steering, power brakes*, etc.) which, when assigned specific values, describe it. Each of these attribute/value combinations contributes to the car's total worth. Each attribute is defined by its name (e.g., a_3 = *engine-size*), its value (e.g., $v_{i,3}$ = *6-cyl*), and its incremental worth (in this case it represents the price) (e.g., $w_{i,3}$ = *500*). In some cases, the difference between cars is in the values of their attributes (e.g., 4-cyl vs. 6-cyl engine-size). In others, it is in whether they even exist (a car has or does not have air conditioning).

The relative versus incremental worth of each attribute can be explained in the difference between buying a new car and a used car. In a new car purchase, the final price of the car is (ideally!) determined by the car's cost of manufacture plus the overhead and profit set by the manufacturer, plus that of each option ordered by the buyer. Reduction in any of these directly affects the final price of the car. In a used car purchase, however, the price of the car is set by the market, rather than by the manufacturer's cost. While increments to the retail price are suggested because of things like low mileage or year of make, these do not add up to the total price of the car. Thus, these represent incremental values.

The Difference-Based Induction Algorithm and Associated Procedures

The basic concept of our approach is that by isolating the differences (as represented by attribute/value combinations) between nearly similar aggregate sets in a

database, the incremental or relative worth of the differentiating attribute/value combination can be said to account for the difference in the total worth of the aggregate sets. The average incremental worth of an attribute/value combination can then be computed from several specific computations of these differences in total worth. We propose a technique that can efficiently identify the differences between aggregate sets that have multiple attributes, and quantify these differences. This quantification of differences is not difficult when the attributes are few, but it can be a daunting task when they are numerous. Thus, the DBI algorithm presented here is designed to segregate aggregate sets in terms of their differences, and quantify the worth of these differences.

The DBI algorithm builds a decision tree whose leaf level nodes each contain a group of identical aggregate sets. In many ways, this approach is similar to ID3 and its successors [Quinlan 1983, 1986, 1987, 1993]. Quinlan's algorithms induce a classification tree from a set of training examples that are described in terms of a collection of attributes, where each example belongs to one of several mutually exclusive classes. A set of rules is produced from this tree that is capable of classifying examples similar to those from which the rules were induced. Each rule is represented by a path in the classification tree that goes from the root node to a leaf node indicating a specific classification. Each level in that path specifies one attribute and the branches emanating from the nodes in a level represent their possible values.

The differences between the DBI algorithm and ID3, C4, C4.5 and other similar ones, are not so much in how each builds the classification tree, but rather, in how the tree is used after it is built, and the source of the examples.

Objective of the DBI Algorithm

In order to understand how the DBI algorithm works, it is important to first understand what it attempts to achieve in the context of the definitions provided in the previous section. The objective of the DBI algorithm is to calculate a general estimate for the incremental worth of each attribute/value combination when compared with another value for the same attribute. More specifically, it empirically computes the average incremental worth, $Iw_{j,x,y}$, of a specific value pv_x of attribute j when compared with value pv_y for the same attribute. This estimate can then be used to estimate the total worth of a new aggregate set when it is compared to other aggregate sets in the database which differ by this attribute/value combination. Ideally, the empirically discovered $Iw_{j,x,y}$ is constant for all aggregate sets, but this is not a necessary condition.

It should be noted that the DBI algorithm does not attempt to determine the individual values of $w_{i,j}$ for each aggregate set in the database. While these values are typically unknown, they are often not necessary, and thus, it is not our main objective. They can be easily computed by making some simple modifications to the algorithm. However, we have left this as a subject of future work.

Description of DBI Algorithm

Each level in a classification tree, as in conventional induction algorithms, represents one attribute. The exception to this is the leaf level, which simply acts as a repository of aggregate sets to be used in further analysis. With this exception, all nodes at the same level represent the same attribute. Each branch emanating from a node represents a specific possible value or a "discretized" range of values for that attribute.

Aggregate sets found in DB are *distributed* throughout the tree according to their attributes and their values. Beginning with the root node, distribution takes place by repeatedly passing each aggregate set from a present node at one level to a child node at the next level through the branch corresponding to the aggregate set's value for the attribute represented by the parent node. Therefore, all aggregate sets collected in any one node are identical insofar as the attributes considered in that level and in all ancestor levels of the tree. As new successor levels are added and the aggregate sets are further distributed, they will be further segregated, with fewer and fewer sets populating any one node. This process continues until all attributes have been represented in the tree up to and including the *critical attribute*, which is represented at the level above the leaf level.

Aggregate sets found at the same leaf node are identical to each other. They, however, are similar to the sets in a sibling leaf node in all attribute values except one - the attribute at the level of the leaves' parent node. Thus, any differences in worth between aggregate sets in sibling leaves can be inferred to be attributed to the difference in the value of that single attribute.

The classification of the groupings to which aggregate sets in the leaf nodes belong is not important in the DBI algorithm. It is only important to know that the sets are similar among themselves, and minimally different from sets populating sibling leaf nodes. Identifying and quantifying these differences is the final objective of the DBI algorithm. The following section describes the technique proposed in greater detail.

Construction of the DBI Algorithm Tree

This section describes the procedure used in the construction of the DBI algorithm tree. This procedure is composed of the following steps: 1) data preparation and preprocessing, 2) selection of critical attribute, 3) tree construction, 4) aggregate set pruning, and 5) paired leaf analysis. The DBI algorithm is described in this section.

Data Preparation and Preprocessing. This step ensures that all the aggregate sets to be used as "training examples" are similar in structure.

Selection of the Critical Attribute. The first step in the DBI algorithm is to select the critical attribute for each of the classification trees to be built. The critical attribute is an important one, as it is the one whose incremental worth is to be induced from the data. To determine the incremental worths for the values of more than one attribute (as would typically be the case), distinct classification trees are built by the DBI algorithm, one for each attribute whose worth for various values is to be determined.

Tree Construction Procedure. The DBI algorithm builds the tree one level at a time, starting with the root node. The number of distinct values represented by the aggregate sets at the current node determines the number of branches to be created at that node. If there is at least one aggregate set supporting a branch, one will be created for that aggregate set. Only the branches that are supported by aggregate sets are incorporated into the node; thus no "null" or redundant branches are ever created using this technique.

The attribute assigned to the root node can be any attribute except the critical one. However, in the interest of efficiency, some heuristics can be used to select the attribute represented at the root node or any other non-leaf node. Please refer to (Daroszewski, 1995) for a description of these heuristics.

Pruning Heuristics. After a complete classification tree is constructed for each critical attribute, the algorithm enters into the *case pruning phase*. Here, heuristics are applied to identify and discard those aggregate sets whose worths are not consistent with those of other aggregate sets in the leaf-level group. See (Daroszewski , 1995) for more details.

Computation of Incremental Worth - The Paired-Leaf Analysis. The final phase of the DBI algorithm is the *incremental worth calculation*, the process of determining the worth for the critical attribute of a DBI tree. This process involves a technique referred to as *paired-leaf analysis*.

Paired-leaf analysis is applied to two sibling leaf nodes. All aggregate sets populating these leaves have identical values for all attributes except for the critical attribute (the parent nodes of the leaf-level nodes). Therefore, it can be inferred that any difference in worth between aggregate sets in two sibling leaves is the difference in their values of the critical attribute.

The paired-leaf analysis, therefore, computes the incremental worth $W_{j,x,y}$ (the incremental worth of the j^{th} attribute having value pv_x versus having value pv_y) by examining $W_{j,pvx}$, the average of the overall worths of the aggregate sets populating a specific leaf (having value pv_x for the leaf node value) with $W_{j,\ pvy}$ the average worth of the aggregate sets populating a sibling leaf (having the value of pv_y for the leaf node value).

For discrete values, the incremental worth is

$$W_{j,x,y} = (W_{pvx} - W_{pvy})$$

For continuous value attributes that have been mapped to intervals, the underlying continuous values should be used, rather than discrete interval names. For these values, the difference between the averages should be further divided by the difference between the values themselves. The formula should now look as follows:

$$W_{j,x,y} = (W_{pvx} - W_{pvy}) / (pvx - pvy)$$

For example, suppose continuous values for automobile acceleration times, which range from 1 to 20 seconds, have been mapped into two-second intervals {[1,2] [3,4] [5,6], ..., [19,20]}. Thus, if one car has an accceleration time of 6 seconds (interval [5,6]), and another 13 (interval [13,14]), the difference used should be (13 – 6) = 7, not the fact that they are 4 intervals away

Finally, the same average difference found between other pairs of sibling leaf nodes could be used to arrive at a final average of the incremental worth for that critical attribute.

Tree Construction Algorithm. The DBI algorithm will organize the aggregate sets in the database in the form of a classification tree. The tree construction algorithm is as follows:

---

1. Read all aggregate sets from the database into memory.
2. Select the attributes whose incremental worth is to be determined from the set of all possible attributes occurring in each aggregate set. It will typically be all, but the option exists to determine the worth of only a subset of the attributes.
3. Put these attributes in a list and call it **Total_Attribute_List.**
4. Make a list called **Critical_Attribute_List** and set it initially to **Total_Attribute_List.**
5. Until **Critical_Attribute_List** is empty, do the following:
   a) Remove the first element of the **Critical_Attribute_List**, and store it in a variable **critical_attribute.**
   b) Make a list called **Test_Attribute_List** and set it to **Total_Attribute_List.**
   c) Remove the **critical_attribute** from the **Test_Attribute_List.**
   d) Create a classification tree consisting of an unlabeled root node.
   e) Until the **Test_Attribute_List** is empty, do:

i) Using the criteria above, select the next test attribute from the **Test_Attribute_List.**

ii) Assign the test attribute to the current node of the classification tree.

iii) Create branches corresponding to possible values of the currently selected test attribute.

iv) Distribute each aggregate set into the next level by assigning them to the corresponding branches, according to their value for the attribute being tested at the current node.

v) Delete the test attribute from the **Test_Attribute_List.**

vi) Create a new unlabeled node level at the end of branches generated.

f) Assign **critical_attribute** to the last node level created.

g) Create branches corresponding to possible values of the currently selected test attribute.

h) Distribute each aggregate set into the leaf level by assigning them to the corresponding branches, according to their value for the attribute being tested at the current node.

i) Prune the aggregate sets irrelevant for analysis through the heuristics described in Section 3.3.4

j) Apply paired-leaf analysis to determine the partial worth estimate for the critical attribute.

k) Compute the final worth for the critical attribute.

6. End

A more detailed description of the system in pseudocode can be found in [Daroszewski, 1995].

Implementation and Evaluation of the Difference-Based Induction Algorithm

A prototype was built which implements the DBI algorithm. The domain used for the prototype was in residential property appraisal. This domain lends itself very well to this approach because determining the incremental worth of the specific attributes of a house (the aggregate set) has a significant impact on its appraised value (its overall worth). Large databases of sold houses are typically available to be used to determine the incremental worth of several attributes.

We developed a DBI algorithm prototype and extensively tested it on a database of 84 single-family houses sold during 1994 in the Alafaya Woods subdivision, Oviedo, Florida. Sales data was obtained from the Multiple Listing Service (MLS) database

The prototype will accept the entry of a feature of a house whose incremental worth is to be computed from the MLS database for that section of the city at that time. It will return a single number indicating what the market considers to be the incremental worth of that property feature when compared to one of lesser worth

Results of Prototype Evaluation

To evaluate its effectiveness, its results were compared to those developed manually by an expert for the same neighborhood area during the same period of time. The attributes compared were the number of bedrooms, the number of bathrooms, the living area, existence of a pool, existence of a garage, existence of a fireplace and age of house. The DBI prototype results were in the form of a range of values, something typically done in the appraisal business. The percent difference between the maximum and minimum of said ranges when compared to the corresponding ranges computed by the expert were as follows (the comments in parentheses represent the expert's evaluation of the comparison): Living area: 0-2.4% ("acceptable"); Bedrooms: 49-72% ("marginally acceptable"); bathrooms: 154-186% ("unacceptable"); garage: 0.3-49% ("acceptable"); swimming pool: 2.5-19% ("acceptable"); fireplace: 2.5-109% ("low end acceptable, high end not acceptable"); age of house: 20-42% ("acceptable"). The complete data and a more detailed discussion of results can be found in (Daroszewski, 1995).

Conclusion

The results obtained indicate that the procedure worked well with a relatively small database. However, unacceptable results were obtained for the number of bedrooms, fireplace, and the number of bathrooms. These discrepancies could be attributed to the limited number of houses populating the leafs used in the paired-leaf analysis.

References

Daroszewski, S.G. 1995. Mining Metric Knowledge from Databases Using Feature-Oriented Induction. Master's thesis, Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL.

Quinlan, J. R. 1983. Learning Efficient Classification Procedures and their Application to Chess End Games. In Michalski, R. S., J. G. Carbonell, and T. M. Mitchell, eds *Machine Learning: An Artificial Intelligence Approach.* San Mateo, California: Morgan Kaufmann.

Quinlan, J. R. 1987. Decision Trees as Probabilistic Classifiers. In Proceedings of the Fourth International Workshop on Machine Learning, 31-37. Irvine, California.

Quinlan J. R. 1993. *C4.5: Programs for Machine Learning*, San Mateo, California: Morgan Kaufmann,.