

On data mining Tree structured data represented in XML

Andrew N, Edmonds
Scientio, Incorporated.
Haydon House, Station Rd,
Woburn Sands, Bucks MK17 8RX UK
e-mail: andy@Scientio.com

Abstract: The ubiquitous nature of XML as a data interchange format and the arrival of XML databases offers new opportunities and challenges to data mining. The XML format permits the representation of data structures that cannot be easily mapped into a relational framework. New opportunities exist to mine relationships expressed by tree position and the presence or absence of sub trees as well as the conventional categorical and numeric data values contained at leaf nodes. This paper will consider why most conventional data mining algorithms are ill suited to mining tree data, describe a novel XML based knowledge representation methodology called Metarule, and describe algorithms that do perform well. Two examples drawn from business will be used to demonstrate the application of these novel techniques.

Keywords: Data-mining, XML, Unstructured data, Metarule, Valuation

1. Introduction

In the course of developing skills in computer programming, a student usually begins by using simple one dimensional arrays, then multi dimensional and thereafter more complex structures like trees and linked lists. It seems the development of data representation and databases is following the same pattern. And just as the requirement for more complex data structures is driven by the inefficiency of simple arrays for the more interesting types of data to be represented, so the world at large seems to be moving finally from relational databases to more expressive forms such as XML and Object Oriented databases. This is not to say that relational databases are doomed, more that there are increasing examples of applications that do not map efficiently to the relational format, no matter how many references and links are permitted.

Ultimately, as in everything else, economics will drive the take-up of new database forms, and if the complexities of developing and supporting relational systems outweigh those of their competitors, the obvious will happen.

XML [5] is an international standard for representing data in a generic fashion. XML is widely accepted as the “lingua franca” of disparate computer systems and forms the basis of B2B initiatives such as the Biztalk consortium, and web services based on the SOAP standard. XML represents data as a tree, and makes no requirements that the trees be balanced; indeed XML is remarkably free form, with the only requirements that there be only one root node, and that nodes can contain attributes, textual data items and child nodes, but not both text and children.

Any data types that can be represented as text, can be included in an XML document, and a matching document called the schema is often supplied to act as the definition of and key to the structure of the first document, and may define the data types of the attributes and text items. Another key component is a method for accessing and addressing elements in an XML document. This is provided by the XPath [8] standard, which has drawn its inspiration from the file and directory addressing model of Unix.

A schema then defines the structure and data types of a document in XML, and of course this is usually a one-to-many relationship. An XML document can hold data in a wide variety of tree structures, subsuming tabular data representations, and everything can be accessed via the XPath standard. To add to the flexibility and complexity of this mix, multiple schemas can govern any

particular document. They can each be mentioned in the document, and namespaces are defined to identify which document elements belong to which schema.

This facility permits the generation of composite documents holding information from multiple sources. So, for instance a company might send another an order containing financial information governed by one schema, and the engineering spec of the parts ordered governed by another, both combined in the same document.

2. XML databases

XML databases are designed to hold XML documents in such a way that they can be easily retrieved at the document level, and node level. They generally support XQuery [7], which is XPath with database handling extensions, and are generally arranged hierarchically, so the database administrator will typically set up “collections” containing multiple documents connected in some fashion, and to aid indexing the collections will also hold the schemata governing the documents in that collection. XML databases offer the same maintenance systems as conventional databases.

3. Problems with extracting knowledge from XML

So, given that large amounts of data are being stored and transferred in XML, how can we go about data mining it, and predicting, classifying and clustering it? The obvious answer, “do as we’ve always done”, does not work for several of reasons.

Firstly, data mining is generally directed at tabular data. Although structures other than tables can be mined, using specialized techniques For instance some forms of web mining, a generic solution does not exist, or has not until now for tree structured data. Tabular data can easily be represented in XML, but not all data that can be represented in XML can be easily represented in a single densely populated table.

Secondly, because there is no requirement in XML that trees be symmetrical, or that sub trees contain equal numbers of children, if a table of learning vectors is assembled from elements within an XML document to hold data, it is likely to be sparse, that is large tracts of it may contain no data. Of the various data mining algorithms available, few can handle this kind of input.

Thirdly, because XML data is hierarchical, the position of a data item within a tree can be relevant. For instance, if analyzing a genetic database arranged as a family tree, both the presence of a genetic characteristic in one individual, and any characteristics found in the parents or grandparents are potentially relevant to an analysis. We will later show examples of where the “tabularization” of data that started out as tree structured has thrown away useful knowledge that must then be inferred. So in addition to the values taken by data items there is the new element of position within a hierarchy to be mined.

So to sum up, any supervised or unsupervised learning algorithm used to mine XML data must perforce make use of example vectors, however derived, but these vectors are liable to be sparse. The presence or absence of data within a particular location in the learning vector is in itself something to be mined, since it may represent the presence or absence of a sub tree, as is also the data values and structure associated with ancestor or descendant nodes.

4. Representing Knowledge in XML

The homomorphism between an XML document and parse trees offers interesting opportunities. A sister specification to that of XML is XSLT, a mechanism to transform an XML document into another XML document or a different form. Typically this is used to display XML data in HTML. It is possible to store knowledge encoded in a parse tree format, i.e. partially interpreted, and to display this in a more human friendly form by use of a simple transform.

The language Metarule, which is defined as an XML schema and is in the public domain [1] was written by the author to take advantage of this. Fundamental to this language is the assumption that knowledge in the domain of interest can be expressed in rules or expressions of the following forms:

(1) If <conditions> then <output > will be <classification>

- (2) If <conditions> then <output> will be <fuzzy set>
 (3) If <conditions> then <output > will be <arithmetic expression>

The basis for the above is fuzzy control theory, where the rule types, and inference mechanisms based on them are described as Takagi, Mamdani , and Sugeno [2] respectively.

A metarule document is composed of a series of such rules, and a data dictionary section describing the types of the inputs and outputs to the rule set.

The <conditions> above can be an expression making use of the usual range of logical operators, fuzzy sets, comparison operators, and arithmetic operators.

Each rule is a mapping between some subspace of the input domain and the output domain, The output domain can be categorical, as in rule type (1), or numeric as in rule types (2) and (3). Rule type (3) can also be used for Bayesian networks.

The language can be read directly by a Metarule inference engine, and data can be applied, either via a programmatic interface or alternatively from an XML document.

A thorough description of metarule is outside of the scope of this paper, but the following shows a solution to the Fisher's Iris data classification problem [11], both as raw Metarule, and transformed via a simple XSLT transform to text.

<pre> <rule> <if></if> <is> <input>petal_length</input> <set settype="small"></set> </is> <then></then> <output>class</output> <willbe></willbe> <category>Iris-setosa</category> <confidence>1</confidence> <examples>1</examples> </rule> <rule> <if></if> <and> <is> <input>petal_length</input> <set settype="medium"></set> </is> <is> <input>petal_width</input> <set settype="medium"></set> </is> </and> <then></then> <output>class</output> <willbe></willbe> <category>Iris-versicolor</category> <confidence>1</confidence> <examples>1</examples> </rule> </pre>	<p>Training set performance 97.36842105263157</p> <p>if petal_length is small then class will be Iris-setosa (confidence 1.00)</p> <p>if petal_length is medium and petal_width is medium then class will be Iris-versicolor (confidence 1.00)</p> <p>if petal_length is medium and petal_width is large and sepal_width is medium or sepal_width is small then class will be Iris-virginica (confidence 1.00)</p> <p>if petal_length is large and sepal_width is small and petal_width is large or petal_width is medium then class will be Iris-virginica (confidence 0.83)</p> <p>if petal_length is large and sepal_width is large or sepal_width is medium then class will be Iris-virginica (confidence 1.00)</p> <table border="1" style="width: 100%; border-collapse: collapse; margin-top: 10px;"> <thead> <tr> <th colspan="3" style="text-align: center;">inputlist</th> </tr> <tr> <th style="width: 20%;">Name</th> <th style="width: 20%;">Type</th> <th style="width: 60%;">Category/Set</th> </tr> </thead> <tbody> <tr> <td>petal_length</td> <td>numeric</td> <td>small 1.000,1.500,4.400 medium 1.500,4.400,5.600 large 4.400,5.600,6.900</td> </tr> <tr> <td>petal_width</td> <td>numeric</td> <td>small 0.100,0.200,1.300 medium 0.200,1.300,2.000 large 1.300,2.000,2.500</td> </tr> <tr> <td>sepal_length</td> <td>numeric</td> <td>small 4.300,5.000,5.800 medium 5.000,5.800,6.700 large 5.800,6.700,7.900</td> </tr> <tr> <td>sepal_width</td> <td>numeric</td> <td>small 2.000,2.700,3.000 medium 2.700,3.000,3.400 large 3.000,3.400,4.400</td> </tr> </tbody> </table> <table border="1" style="width: 100%; border-collapse: collapse; margin-top: 10px;"> <thead> <tr> <th colspan="3" style="text-align: center;">outputlist</th> </tr> <tr> <th style="width: 20%;">Name</th> <th style="width: 20%;">Type</th> <th style="width: 60%;">Category/Set</th> </tr> </thead> <tbody> <tr> <td>class</td> <td>categorical</td> <td>Iris-setosa Iris-versicolor Iris-virginica</td> </tr> </tbody> </table>	inputlist			Name	Type	Category/Set	petal_length	numeric	small 1.000,1.500,4.400 medium 1.500,4.400,5.600 large 4.400,5.600,6.900	petal_width	numeric	small 0.100,0.200,1.300 medium 0.200,1.300,2.000 large 1.300,2.000,2.500	sepal_length	numeric	small 4.300,5.000,5.800 medium 5.000,5.800,6.700 large 5.800,6.700,7.900	sepal_width	numeric	small 2.000,2.700,3.000 medium 2.700,3.000,3.400 large 3.000,3.400,4.400	outputlist			Name	Type	Category/Set	class	categorical	Iris-setosa Iris-versicolor Iris-virginica
inputlist																												
Name	Type	Category/Set																										
petal_length	numeric	small 1.000,1.500,4.400 medium 1.500,4.400,5.600 large 4.400,5.600,6.900																										
petal_width	numeric	small 0.100,0.200,1.300 medium 0.200,1.300,2.000 large 1.300,2.000,2.500																										
sepal_length	numeric	small 4.300,5.000,5.800 medium 5.000,5.800,6.700 large 5.800,6.700,7.900																										
sepal_width	numeric	small 2.000,2.700,3.000 medium 2.700,3.000,3.400 large 3.000,3.400,4.400																										
outputlist																												
Name	Type	Category/Set																										
class	categorical	Iris-setosa Iris-versicolor Iris-virginica																										

Figure 1, Example of metarule solution to Fisher's iris data and transformed human friendly version

Metarule has been designed to support a wide range of knowledge representation methodologies, and has been used to represent fuzzy logic expert system rules, neural networks and Bayesian belief networks successfully. Once encoded in metarule, the inference engine can be used to exploit the rule set whatever the methodology created it.

5. Mining XML

Given that we have source data in XML, governed by one or more arbitrary schemata, and that we wish to represent the knowledge we discover in Metarule, how do we go about mining the data?

The first problem is to define what we wish to mine. We may wish to predict the value of some data element in the data source, to classify groups of elements in the data source, and to predict the presence or absence of sub trees within a data source.

To perform the above we must select data items within the data source to act as inputs to the predictive process.

Furthermore we must attempt to find enough learning examples within our data source to offer our learning algorithms some hope of working.

The solution chosen is to identify a set of internal, i.e. not terminal, nodes within an XML document, using an XPath expression, and consider each node as representing a learning example. Data items that are used as part of the learning example vector must be in nodes accessible relative to those learning example nodes, normally terminal nodes beneath them. Because of the free form nature of XML each learning node may have a completely different mix of child nodes. This is what makes data mining XML difficult.

We identify each of the data items using XPath expressions relative to the learning nodes. The following diagram illustrates the concept.

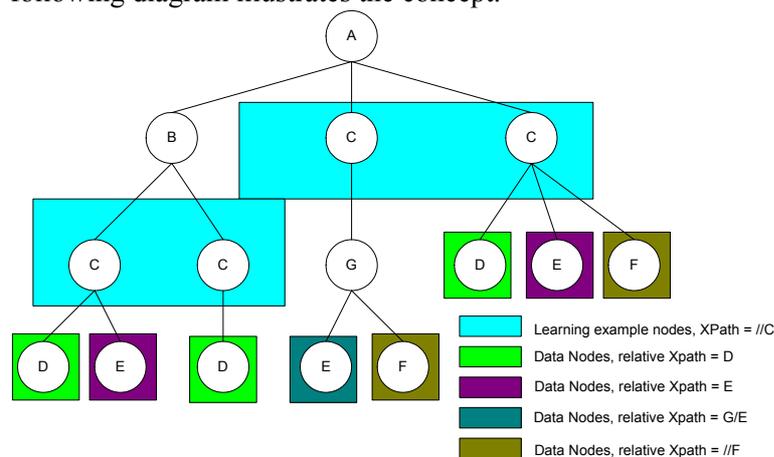


Figure 2, Addressing Learning example and data nodes using XPath

We can use the same mechanism to identify both inputs and outputs, if supervised learning is employed. We can access with our data item XPath expressions both internal and terminal nodes. The latter can be tagged as numeric or categorical depending on data type and purpose, the former we can detect as being present or absent. XPath is so designed that elements can be addressed both above and below our learning nodes, so the kind of hereditary data mentioned earlier can be easily accessed.

Given the requirement that the learning vectors can be sparse, and that the gaps in the vectors may occur in different places for each vector, our choice of learning algorithms is limited. In this implementation supervised learning algorithms have been used, though the methodology can be extended to unsupervised learning. An coding layer could be generated for Neural Networks to be used, but given our requirement to encode the result in Metarule, and given the desirability of creating rules that can easily be understood, it was decided to use Fuzzy Logic Tree induction. This particular

methodology follows the work of [10] and is conceptually similar to [3] although developed independently.

6. Data preprocessing

The first stage of processing is to discover the characteristics of the input data. In the software used to evaluate these techniques, the name, type and XPath expression for each input and output is supplied via an XML document, as well as the generic XPath expression for selecting the learning examples. A wide variety of XML parsers exist, following two specifications, DOM [6] and SAX. In this implementation the former was used. Applying the generic XPath expression to the parsed version of the source data via the parser results in a set of nodes each corresponding to a learning vector. The application of the input XPath expressions to each of these yields arrays associated with each input of data values. As discussed earlier, these data values could be categorized as *categorical*, *numeric* or *presence*. If the former a table was generated containing each unique instance in the data set, if *numeric* triangular fuzzy sets were generated, so arranged that equal numbers of input points would fall into each partition, if *presence*, the presence or absence of data was noted as a Boolean variable. As is normal practice, the fuzzy set shapes in the case of a numeric variable are so arranged that for any point in the universe of discourse the sum of set memberships will be 1. In this implementation the number of outputs that can be defined is limited to 1.

7. Fuzzy Decision Tree generation

The decision tree generation used in this work is a generalisation of Quinlan's ID3[4]. The pre-processing stage transforms the data to a logical domain, where logical rules can be constructed. Unlike ID3, which partitions numeric variables using cut points, this algorithm operates on the set membership values. Thus *numeric* variables are partitioned into a user selected number of sets, *categorical* variable are partitioned into the different discrete categories discovered in the data, and *presence* variables are partitioned into two categories, present and absent.

- 1) The algorithm proceeds by first forming the set of learning vectors and creating the top-level decision tree node.
- 2) The information gain, I , is calculated for each input relative to the output for each learning vector. The implementation is arranged to ignore learning vectors where either of the data elements considered are empty.
- 3) The input corresponding to the greatest information gain exceeding a constant is selected. This input is associated with the current decision tree node.
- 4) Now the set of learning vectors is partitioned into n sets where n is the number of partitions defined for that input according to that inputs membership of the partitions. Decision nodes are created for each partition, associated with the partitioned data and linked as children of the current node.
(Note that this is a crisp rather than fuzzy partitioning of the data.)
- 5) Finally the algorithm is called recursively from (2) on each child node.

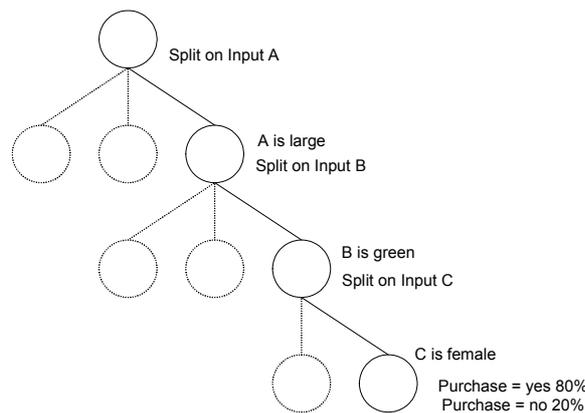
A leaf node is declared when the number of elements in the learning vector set has dropped below some pre-determined constant, or when none of the information gains have exceeded a threshold. In order to avoid overtraining and encourage parsimony the threshold is calculated as $a + bp$ where a and b are constants and p is the distance of the current node from the root.

8. Creation of a rule set from the decision tree

Having created the decision tree, the next stage is to label the leaf nodes with respect to the output. Each leaf node has associated with it a set of learning vectors. The sum of the memberships of each vector over the partitions of the output are calculated and then normalised by being expressed as fractions of the sum over all the memberships.

From any leaf node, the path to the root represents the logical product of the input/ partition combinations that created it.

Rules can therefore be generated by taking each output partition represented with a membership sum above or equal to a threshold and reading back to the root. A recursive algorithm exists to do this simply.



Rule: If A is large and B is green and C is female then purchase will be yes (80%)

Figure 3, Converting decision trees to rules

The threshold is set to be $1/(\text{number of output partitions})$.

The story is not quite over there since there can be considerable inefficiency in the rule sets produced. Many rules differ in only the last term to the left of the “then”, and so can be combined together, data type permitting. Categorical and presence variable can be freely combined; numeric variables can only be combined if the fuzzy sets used are contiguous.

If a is black and b is white and c is yellow then output is true
If a is black and b is white and c is green then output is true

Becomes:

If a is black and b is white and (c is yellow or c is green) then output is true

The recursive algorithm can be modified to perform the above.

One further, and vital refinement is the association with each rule of a confidence figure set to the normalised membership sum.

The rule set created and input and output definitions are then encoded in Metarule.

9. Evaluation of rule set and testing

The final stage is to evaluate the rules independently on the Metarule inference software and calculate performance. The current implementation permits a randomly selected sample of data to be set aside for testing. The user can set the percentage for this.

10. Metarule inference implementation

There are some aspects of the Metarule Inference software that need to be clarified.

The metarule inference software reads metarule and creates an internal representation of the rules set, and of the inputs and outputs specified. The rules are implemented directly as parse trees with the same structure as the Metarule source.

Logical operators are coded as the fuzzy logic variants of Boolean operators, that is to say truth-values are real numbers in the range (0,1) and the operators behave as Boolean operators at the extremes of the range.

10.1 Handling null or unknown inputs

Our implementation also includes the representation of an unknown state, represented as -1 , and the operators treat unknowns rationally. So for instance if either or both of the inputs to an “and” function are -1 , the output is -1 . If one of the inputs to an “or” function is -1 , the output is set to the value of the other input.

10.2 Representing uncertainty

Metarule supports various representations of uncertainty. As discussed above, the absence of knowledge of the state of an input value can be handled. If vague information is known, numeric inputs can be represented as fuzzy numbers from simple intervals, triangular sets expressed as triplets, and trapezoidal sets described by four values. The uncertainty information is retained in the outputs, where, as well as providing a central prediction, with associated confidence value, alternate ranked categorical results or triangular set values are presented.

10.3 Rule evaluation

Evaluation of a rule is conceptually a bottom up process where first input values are instantiated, then compared to the appropriate category, set definition, or merely checked for non-null status in the case of categorical, numeric and presence types respectively. The output of these processes is a degree of truth-value in the range (0,1) with the special case of -1 representing unknown. These values are propagated up through any logical operators till the root of the rule is reached. This determines the left hand side of the rule’s activation.

10.4 Aggregating the results of multiple rules

We have not yet considered the aggregation of the outputs of multiple rules, should more than one be activated.

In all but the simplest applications this is a distinct possibility, most real world processes and the data collected from them exhibit some degree of vagueness which will engender multiple and possibly contradictory rules.

For each rule that fires, i.e. has an activation value > 0 , a firing strength value is calculated as the product of the activation and the rule’s confidence.

Aggregation is achieved in a variety of ways depending on the type of the output value. Outputs can be categorical, in which case the possible output categories are ranked in descending order of rule’s firing strength. If outputs are numeric, and the rule is type (2) as defined previously the rule will contain a fuzzy set definition. (Fuzzy sets, you will remember are defined for all numeric inputs and outputs during the pre-processing stage.) The resulting activation is calculated using the centre of gravity algorithm. Fuzzy arithmetic, based on Zadeh’s extension principle [12] is used throughout, so the output is a fuzzy number. The same process is used with type (3) rules once the output expression is evaluated.

Finally the learning vectors are applied sequentially to the Metarule Inference software and appropriate error measures (hit rate or sum squared error) are calculated.

If part of the learning set was put aside as a test set before training began, this is then separately evaluated. For categorical outputs the facility exists to specify costs and these will be calculated at the same time. The Metarule rule set is output as a file and retained for further use with the Metarule Inference software.

11. Applications

11.1 Valuation works of art at auction

The source data for this example is 640, 000 auction records representing sales from auction houses over a ten year period. The data consists of records of sales, listing when, what was sold, how much in a variety of currencies, the medium the art was created in, sizes and the artist's name, dates and nationality.

The task was to create a software application so that a user could input an artist's name, the medium of a work and the sizes, and receive an approximate valuation.

With a minimal knowledge of the art market one might guess that the key influence on the value of a work of art is the artist who painted it. Next comes condition, the content of the painting, then medium and size. Since the records came from reputable auction houses (hopefully not an oxymoron!) the attribution was not in doubt, but the records give no indication of condition and subject matter, we could not for instance differentiate landscapes from portraits. The next problem is that there is no unique identifier for artists. Names are sometimes misspelled and the dates could be exact from some sources, and merely say "19th century" from others.

When predicting value, the fact that the values of art at auction had varied dramatically over the ten years that the records were collected, with changing taste and state of the economy in general added a further complication.

The first task was to create a database of artists and assign the pictures to them.

A "data cube" of price, size, medium, and year of sale was constructed, breaking each variable into ten intervals, and normalisation values were calculated to create a normalised price set to the last year of the data records, 1996.

The vast majority of the auction records related to oil paintings, water based media and drawings. An artist may have created works in any combination of these media. A rating based on the ratio of the artist's price achieved for a given medium and size to the average for that size was generated. The rating was averaged over the works for each medium creating a rating for that artists work in each medium. If an artist did not create works in a particular medium the rating would be empty.

Finally a training set was generated where each record represented an auction record, containing rating information, the medium of the work, the width and height and normalized price. Because of the huge amount of data available, only 5% was used, randomly selected, for the training. The resulting rule set was tested on the entire set of auction records, however. All numeric data items were represented with 9 fuzzy sets. Because ratios were not necessarily available for all media for a given artist, and because of the generally noisy and incomplete state of the source data, this data set formed a good demonstration of the ability of the techniques described to cope with incomplete data.

Valuations of paintings are commonly presented as ranges. Typically valuers will use the same logarithmic pricing system used in auctions, although with wide bands. Values tend to be limited to 3 per decade, so for instance 100-300, 300-600, 600-1000. Our system was limited to 9 sets, chosen, as you will recall, so that an equal number of points fell into each set. very few items were sold for less than \$500, and very few for more than 50,000 so our fuzzy sets were at least comparable in size of band to those used by professional valuers.

Other sources [13] have found that professional valuers are correct only 33% of the time, with a further 33% above and below. This methodology was correct 51% of the time on the entire data set, with 26% higher and 23% lower. One can only speculate why our system would outperform valuers, who have the benefit of actually seeing the pictures.

11.2 Analysis of an Organization Chart

An interesting example of tree shaped data that might yield useful information is an Organization chart, especially if it contained numeric data such as age, salary etc. Unfortunately it was not easy to source this information from a real large organization, so it was decided to construct a dummy data set.

The form of the dummy data is determined by the following schema:

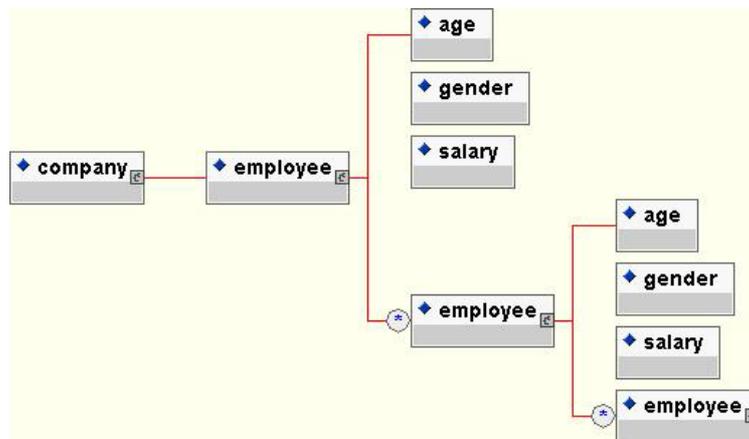


Figure 4, Schema for organization chart dummy data

The tree was constructed recursively by setting the salary of the root employee to \$100,000 and randomly assigning ages between 21 and 65 and genders with equal probability. Employee nodes were enforced for the first 2 levels from the root, and randomly set at between one and seven nodes. The maximum depth was set at 5. For levels 3 and 4 the presence of child nodes was decided randomly with an 80% probability.

The salaries were calculated for each node depending on the salary at the parent node p , the age a , the gender g , the presence of child nodes c and a random value between 0 and \$100, r . The calculation was set to depend on

$$\text{Salary} = 0.8p + 0.0022*(a - 21)p + 0.1p(\text{if } c) - 0.08p(\text{if } g = \text{female}) + r$$

The learning node selection path was set to `//employee`, thus finding all the employee nodes in the chart, and `salary` was predicted from `age`, `gender`, the presence of `employees` and the employers `salary`, `age` and `gender`. The eventual tree contained 267 dummy employees.

The following rule set was generated from the data, and was found to be accurate to within an RMS error of \$1,324 in sample, and \$1800 out of sample. Salaries in the data set ranged from \$32,000 to \$100,000 with an average of \$45,750 and standard deviation of \$14,100

12. Conclusions

Data mining tree structured data requires different techniques and offers new opportunities, in that both the value of data items and their environment may contain useful information and be mined. It has been shown that Fuzzy Tree induction has the required characteristics of robustness in the presence of inorthogonal data, and techniques and considerations for successfully mining arbitrary tree shaped structures have been discussed.

13. Citations

- [1] XML.org Schema repository, <http://www.xml.org/xml/registry.jsp>
- [2] Nauck, Klawonn & Kruse, Neuro-Fuzzy systems, Wiley, 1997
- [3] Baldwin & Lawry, A new approach to learning linguistic control rules, International Journal of Uncertainty, Fuzziness and Knowledge based systems.8.1. pp 1-43
- [4] J.R. Quinlan, "Induction of decision trees", Machine learning 1(1986) pp81-106
- [5] XML specification: <http://www.w3c.org/XML/#9802xml10>
- [6] DOM specification <http://www.w3c.org/DOM/>
- [7] XQuery specification <http://www.w3c.org/XML/Query>
- [8] XPath specification <http://www.w3c.org/TR/xpath>
- [9] XSLT specification <http://www.w3c.org/Style/XSL/>
- [10] Ichihashi et al, Neuro Fuzzy ID3, Fuzzy Sets and systems 81 (1996) 157-167
- [11] Fisher, R. A. (1936). The Use of Multiple Measurements in Axonomic Problems. *Annals of Eugenics* 7, 179-188.
- [12] L.A. Zadeh, (1975), The concept of linguistic variable and its applications to Approximate Reasoning, Part I, Information Sciences, 8, 199- 249; (1975), Part II, Information Sciences, 8, 301-357; (1976). Part III, Information Sciences, 9, 43-80.
- [13] Personal communication from M.D. of Art Index Limited, UK