Modelling Large Datasets Using Algebraic Datatypes: A Case Study of the CONFMAN Database

Markus Mottl
Austrian Research Institute for Artificial Intelligence
Schottengasse 3, A-1010 Vienna, Austria
markus@oefai.at

May 15, 2002

Abstract

Being able to provide clear specifications of large datasets comprising hundreds of variables, each of which can take on many different values, while still being able to efficiently and accurately learn functional relations from such data would certainly make data mining techniques even more viable in the real world. In this report we describe a new modelling approach, which essentially generalizes discrete decision tree learning to induction of non-recursive functions over algebraic datatypes. Taking the CONFMAN mediation database as guiding example, we demonstrate how this approach allows us to give more natural data specifications that can take into account semantic aspects which are hard or even impossible to model in common attribute-value representations. We will also explain how this can have a positive impact on accuracy and efficiency.

1 Introduction

The CONFMAN mediation database[1], a detailed collection of currently 3676 mediation attempts in 309 conflicts, consists of a total of 237 attributes including 1168 different attribute values. Achieving improvements in excess of the base line accuracy has turned out to be hard[2], which may be a consequence of the used data representation not allowing the expression of certain semantic properties. Furthermore, the comprehensibility of learnt models may also benefit from more fine-grained representations.

This report will not go about the details of learning using the more expressive representation that builds on algebraic datatypes, but will focus on how to

[1] More information can be found in [BL93].

[2] See [FPT97] for a survey of applying machine learning techniques to this database. A good introduction to machine learning can be found in [Mit96].

design such data specifications, describing their advantages as we go along. It suffices here to mention that learning using the strictly more general representation can be done equally efficiently as with decision trees. The generalized algorithm actually performs absolutely identical when given a specification that can be handled in the traditional way.

We will furthermore assume that the reader has a basic understanding of algebraic datatypes[3], which are conceptually quite simple.

2 Improving existing specifications

Rather than just enumerating improvements to the data representation in our database, we will discuss general principles that can guide us to improved specifications of the usual attribute-value encoding as required by ordinary decision tree learning, but we will do so by providing concrete examples for demonstration purposes.

2.1 Local improvements

2.1.1 Problem: handling ordered values

One immediately obvious shortcoming of the usual representation is the impossibility of expressing order relations between attribute values. This can be seen most easily with discretized numerical values. For example, the variable CM10A in the CONFMAN database tells the number of months that a conflict persisted before intervention by conflict management. There are seven possible values with the following meaning:

value tag	description
0	pre-intervention
1	1-2
2	3-6
3	7-12
4	13-24
5	25-36
6	37+

Table 1: CM10A: number of months before conflict intervention

This encoding drops significant information, namely that these values can be ordered (on a time scale). The learning algorithm is forced to discriminate between all these cases, which means that provided training data is split up into seven subsets here. The more values the variable can take, the smaller those

[3] Also sometimes referred to as *(recursive) sums of products*. See, for example, [Win93] for a formal treatment.

subsets, which will usually impose a strong bias in subsequent learning steps. This is also suggested by empirical evidence in surveys concerning the "small disjuncts"-problem[4].

This problem also shows up in a dual way when we consider target attributes. Instead of just predicting whether some mediation outcome is a success or failure, we might be interested in the exact quality. The attribute CM14 expresses it as one of five values:

value tag	description
1	med offered only
2	unsuccessful
3	ceasefire
4	partial agreement
5	full settlement

Table 2: CM14: detailed outcome of mediation attempt

Here, again, our background knowledge tells us that there is an implicit order hidden in these values. We would certainly expect that, for instance, a ceasefire is a requirement for obtaining a full settlement. This should also change the way we estimate classification accuracy. Predicting "unsuccessful" is surely worse than predicting "partial agreement" if indeed a "full settlement" was achievable.

This problem should not be confused with cost-sensitive learning as described in e.g. [Dom99] or similar techniques that take a detour over regression as in e.g. [KWPD01] to predict ordinal classes. We will show that ordered structures can be represented naturally without the help of numerical values of any kind.

2.1.2 Solution: expressing order with algebraic datatypes

To remedy this shortcoming, we now explain the way in which ordered structures can be expressed using algebraic datatypes. These datatypes are defined as (potentially recursive) type equations. On the left hand side we have the type name and on the right hand side a disjoint union of values, which are separated by bars. Such values consist of a constructor (starting with a capital letter) such that the value is uniquely tagged, and this constructor may take an argument. The argument is either a type or a (Cartesian) product of types. This way we can specify any kind of inductively definable datastructure[5].

[4] See [WH00] for such a survey.

[5] It may be interesting to point out that algebraic datatypes (sums of products) actually form a semi-ring. This also means that they are isomorphic to, for example, context-free grammars. This again implies that this representation is suitable for learning problems in the domain of natural language processing: it can essentially be used to induce (tree) transducers for parse trees originating from context-free languages.

Returning to our previous example, instead of defining the variables as follows:

```
cm10a = PreInt | M1_2 | M3_6 | M7_12 | M13_24 | M25_36 | M37p.
cm14  = MedOff | UnSucc | CeaseFire | PartAgr | FullSett.
```

we could use the following definitions (type equations), which exhibit the underlying ordered structure:

```
cm10a    = PreInt | LateInt late_int.
late_int = M1_2   | M3p m3p.
m3p      = M3_6   | M7p m7p.
m7p      = M7_12  | M13p m13p.
m13p     = M13_24 | M25p m25p.
m25p     = M25_36 | M37p.

cm14    = MedOff    | MedAcc med_acc.
med_acc = UnSucc    | Succ succ.
succ    = CeaseFire | Agree agree.
agree   = PartAgr   | FullSett.
```

Data values would now be represented in a structured way. Instead of encoding, for example, a ceasefire as "`CeaseFire`", we would write

```
MedAcc (Succ CeaseFire)
```

which makes clear that a ceasefire is an accepted, successful mediation attempt.

Conversely, an intervention after two months would be encoded using the expression "`LateInt M1_2`" ("late intervention after 1 or 2 months") rather than as "`M1_2`" directly.

How does this help us? For instance, it might be completely irrelevant for the learner to know that some mediation attempt was undertaken, say, more than six months too late. Therefore, the learner could stop splitting up the dataset according to the number of months exceeding this limit and instead continue with other, possibly more relevant input variables, which is likely to improve accuracy (can choose better variables in subsequent splits) and efficiency (no need to learn more subtrees than required).

For what concerns structured output values, the resulting classifier is allowed to start constructing values even before it is sure about their exact form. For example, given some political scenario, the classifier might be able to predict with very little data that the outcome of a mediation attempt will be successful, but might need much more information to classify the exact kind of success, i.e. a ceasefire only or even a (partial or full) agreement.

This property is also very important from a pragmatic point of view. It may often be extremely costly to collect all relevant data. If mediators are only

interested in whether the outcome will be successful irrespective of the details, they need only collect this much information as required for this goal. On the other hand, if input data about the conflict is already available, we can guarantee that the classifier will already predict every property that only requires this data. In short, we can predict a maximum of information about the result with a minimum of information about the input.

It may also be useful to depict the difference between ordinary encodings and ones that allow subattributes in a visually more appealing way[6]:

cm14				
MedOff	**UnSucc**	**CeaseFire**	**PartAgr**	**FullSett**

Table 3: CM14 represented in the standard way.

<table>
<tr><td colspan="5">cm14</td></tr>
<tr><td rowspan="4">MedOff</td><td colspan="4">MedAcc med_acc</td></tr>
<tr><td rowspan="3">UnSucc</td><td colspan="3">Succ succ</td></tr>
<tr><td rowspan="2">CeaseFire</td><td colspan="2">Agree agree</td></tr>
<tr><td>PartAgr</td><td>FullSett</td></tr>
</table>

Table 4: CM14 represented in a structured way.

2.1.3 Factoring out local information

Taking a general view on subattributes, we see that they are useful for factoring out common information from values which would otherwise be encoded as atomic tags. We now demonstrate this with variable CM18[7]:

value tag	description
1	one medtr
2	two medtrs-same interests
3	two medtrs-diff interests
4	group medtrs-same interests
5	group medtrs-diff interests

Table 5: CM18: number of mediators acting in the mediation event

[6]Types (sets) are written *italic* whereas constructors are written in **bold** letters.

[7]Like value orderings, this pattern is quite frequent in the database (e.g. see also variable CM29, D7, etc.

Instead of forcing the learning algorithm again to discriminate between five values, we can give hints that some of these values have common information by using a more appropriate encoding:

```
cm18 = OneMedtr | TwoMedtrs interests | GroupMedtrs interests.
interests = Same | Diff.
```

Then, again, the learning algorithm need not continue discriminating further if the interests of the parties do not make a difference but their number does. The algorithm is even free to look at the mediators' interests in certain cases, say, if there are only two mediators.

Note that there are not always unique factorizations. For example, we could also have encoded the upper variable as follows:

```
cm18 = OneMedtr | Same medtrs | Diff medtrs.
medtrs = Two | Group.
```

Doing this, we put more stress on the ability of multiple mediators to coordinate their interests rather than on their number. This way we can bias learning by putting discriminating features (value constructors) that we consider more important closer to the top of structured values.

But what do we do if we are unsure about which feature to prefer? Then we can just leave this learning task to the algorithm again and encode our data as follows:

```
cm18 = OneMedtr | TwoOrMore (size * interests).
size = Two | Group.
interests = Same | Diff.
```

Of course, the question about exact size or commonality of interests only makes sense if there are at least two mediators, assuming that mediators cannot be schizophrenic. This example also demonstrates the usefulness of allowing more than one subattribute by employing the *product type operator* "`*`". A group of mediators having differing interests can now be denoted by:

```
TwoOrMore (Group, Diff)
```

2.2 Global improvements

The structure improvements we have considered so far have only affected one variable. By rearrangement of values in a structured way, we could express semantic relations between them.

We can, however, improve existing specifications even more by considering several variables at once and modelling them in a different way. This will most often result in fewer but more structured variables. Due to lack of expressiveness with usual encodings, there are many cases where redundancies exist which can be eliminated.

For example, consider variables P23 and P24:

value tag	description
1	both Christian
2	both Muslim
3	both other global religions
4	both traditional-indigenous
5	both mixed religious base
6	different religions

Table 6: P23: Religion type compared

value tag	description
1	same religion
2	different religion

Table 7: P24: Religion identity of parties compared

As we can see, both variables allow one to decide whether two parties have the same or different religions, but one of them (P23) is much more detailed. This is obviously a workaround invented by the database designers, because they could not express both the abstract and detailed views adequately without redundancies.

The solution here is to merge equivalent values or constructors as follows:

```
pm23_24 = Same religion | Different
religion = Christian | Muslim | OtherGlobal | Traditional | Mixed
```

The learning algorithm would initially see only one variable, which essentially captures the meaning of the variable P23. Only in the case of two parties having the same religion would the algorithm get access to the details as encoded in P24.

This has several advantages:

- Users cannot accidentally (or even intendedly) enter inconsistent data, e.g. that the two parties have differing religions in variable P23 and at the same time that both are Christian in P24.
- Models never contain redundant parts. E.g. the more abstract variable P23 could be highly relevant in general, but P24 may be so in particular for some values only. In such a case a learning algorithm might decide to split up the data according to both P23 and P24 one after the other, which leads to redundant generation of submodels for the "different religion" cases.

- The algorithm can handle the data more efficiently, because it will only get to see the detailed layer once it has matched the abstract layer, whereas it would have to handle not yet relevant data during splits in less expressive representations. E.g. as long as P24 has not been used, it is superfluous work to handle P23.

It is much more effort in general to factor out redundancies among many variables, but this is certainly justified by the benefits.

3 Design hints

In this section we will give a short summary of design hints that one should apply when creating new data specifications.

3.1 Top-down design

The following steps may give a suitable start:

- Identify commonalities among all observable variables and encode this background knowledge by factoring out redundancies using (products of) subattributes (see section 2.2).
- Make sure that bias encoded in the specification truly reflects relations in the real world. If unsure, you may try different representations that leave the task of learning such relations to the employed algorithm as described in section 2.1.3.
- When given several values, identify relations, especially order relations, between them and make use of representation techniques as presented in section 2.1.1.

3.2 Treatment of missing values

An annoying but important point in machine learning concerns the treatment of missing values, i.e. value tags that indicate that some value could not be observed or is even meaningless in a certain context. An often used approach applied in traditional techniques is to explicitly encode ("hardwire") strategies for handling missing values in the learning algorithm. Given more expressive representations it seems necessary to ask whether this is still required or even advisable: more declarative ways of specifying (handling of) missing values may be possible.

Lets consider the following definition:

```
t = A | B
```

If this variable could have missing values, then users of machine learning systems often extend this definition as follows:

```
t = A | B | Missing
```

This, however, may cause not necessarily expected results. Assume that some dataset that uses `t` as result variable contains ten As, nine Bs and eleven missing values. Then an algorithm taking the most frequent value would predict `Missing`. But since the opposite of a missing value is an available value, we would actually have to predict that there will indeed be an observation. This case happens more frequently, because it encompasses both the observations of As and Bs.

Therefore, it may be more suitable to choose the following definition:

```
t = Observed value | Missing
value = A | B
```

This advises the learning algorithm to treat observed values separately from missing ones. We can even specify structures that explain in more detail how to interpret missing values. For example:

```
t = Relevant observable_value | Irrelevant
observable_value = Observed value | Missing
value = A | B
```

This allows more precise specification of why some value is actually missing: because it may be irrelevant[8] or because it could not be observed for some reason, etc. In all these cases the result may indeed be a consequence of the input data, which is why prediction of missing values does indeed make sense. E.g., some value may have been unobservable, because of other (observable) conditions that make collecting data very difficult.

4 Parametric polymorphism for more flexible data specifications

Even though the presented representation allows expressive specifications, there are further extensions that make this task even easier. Modern type systems allow parametric polymorphism, which introduces type parameters for more reusability.

For example, when many variables can have missing values, it would be cumbersome to define, for each of these, a type that wraps around the actual values to tag them as observable as in:

```
t = Observed value | Missing
value = A | B
```

Instead we could write down a more generic definition:

[8] Note that irrelevant values are usually an indicator for redundant information so you might consider reorganizing your data specification.

```
optional value = Some value | None
```

Similar to functions this makes it possible to use abstractions for specification. E.g.:

```
v1 = optional government
v2 = optional religion
```

This results in much more compact and readable data specifications, because it is not necessary anymore to explicitly write down structures that are isomorphic anyway. Legal values in the upper example could be `Some Democracy` of type `optional government` or `Some Muslim` of type `optional religion`. `None` would be legal in both type contexts.

This extension is currently not yet implemented in our system, but is not particularly difficult to achieve either, because the theory behind is well-understood.

5 Conclusion

As we have hopefully succeeded to show, algebraic datatypes give designers of databases for machine learning systems a much broader set of design options, allowing them to model the problem domain more accurately without sacrificing efficiency. Future research may attempt to reveal more ways to benefit from advances in type theory so that even more invariants can be represented in data specifications while still allowing efficient model generation.

6 Acknowledgements

The author wishes to thank Gerhard Widmer for comments on an earlier draft.

This research is supported by the projects "A New Modular Architecture for Data Mining (P12645-INF)", financed by the Austrian *Fonds zur Förderung der wissenschaftlichen Forschung (FWF)*, and "Universities Researching for Society: Peace and the Avoidance of Violence", financed by the Austrian Federal Ministry for Education, Science, and Culture. The Austrian Research Institute acknowledges basic financial support from the Austrian Federal Ministry for Education, Science, and Culture.

References

[BL93] J. Bercovitch and J. Langley. The nature of dispute and the effectiveness of international mediation. *Journal of Conflict Resolution*, 37(4):670–691, December 1993.

[Dom99] Pedro Domingos. Metacost: A general method for making classifiers cost-sensitive. In *Knowledge Discovery and Data Mining*, pages 155–164, 1999.

[FPT97] Johannes Fürnkranz, Johann Petrak, and Robert Trappl. Knowledge discovery in international conflict databases. *Applied Artificial Intelligence*, 11(2):91–118, 1997.

[KWPD01] Stefan Kramer, Gerhard Widmer, Bernhard Pfahringer, and Michael DeGroeve. Prediction of ordinal classes using regression trees. *Fundamenta Informaticae*, XXI:1001–1013, 2001.

[Mit96] Tom M. Mitchell. *Machine learning.* McGraw Hill, New York, US, 1996.

[WH00] Gary M. Weiss and Haym Hirsh. A quantitative study of small disjuncts. In *AAAI/IAAI*, pages 665–670, 2000.

[Win93] Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction.* Foundations of Computing Series. MIT Press, February 1993.