Decision Management Community Challenge September 2020

Decision Table Simplification: Alternative Heuristic Approaches

(Bob Moore, JETset Business Consulting, 15 October 2020)

1 Problem Statement (from the web site)

Using common sense people can compress decision tables. For instance, if you analyze the decision table on the left, you would be able to replace it with the decision table on the right. Instead of 18 rules, we got only 6!

Min Age	Max Age	Card Type	Discount Code
18	30	Standard	0
18	30	Gold	2
18	30	Platinum	3
31	40	Standard	1
31	40	Gold	1
31	40	Platinum	1
41	50	Standard	1
41	50	Gold	2
41	50	Platinum	3
51	60	Standard	1
51	60	Gold	2
51	60	Platinum	3
61	70	Standard	1
61	70	Gold	2
61	70	Platinum	3
71	120	Standard	1
71	120	Gold	2
71	120	Platinum	3

Card Type	Min Age	Max Age	Discount Code
Standard			1
Standard	18	30	0
Gold			2
Gold	31	40	1
Platinum			3
Platinum	31	40	1

In this case, the right table will produce the same results as the left one. However, when a decision table includes much more attributes (columns), the manual analysis becomes difficult or impossible even if you allow a certain level of unsuccessful results. In most cases, you need some special tools provided by digital decisioning products. In this challenge, we ask you to try to compress the following multi-hit decision table:

Type	Adjustment	Adjustment	Loss	Loss	Classified As
					NONE
31	>200		<-150		TOP
31		<200		>=-189	BOTTOM
32	>500		<-1000		TOP
32		<500		>=-99	BOTTOM
33	>500		<-1000		TOP
33		<500		>=-100	BOTTOM
34	>500		<-1000		TOP
34		<500		>=-100	BOTTOM
35	>500		<-800		TOP
35		<500		>=-100	BOTTOM
36	>500		<-800		TOP
36		<500		>=-100	BOTTOM
37	>500		<-2000		TOP
37		<500		>=0	BOTTOM

Please submit your solutions using your favourite tools or just common sense.

2 First Look

The first observation is obviously that we have no context. We do not know where this decision table comes from. This is potentially a problem. If we do indeed 'simplify' the decision table, we may make it difficult to maintain. Simplification implies identifying patterns in the rules, but with no context, we cannot be sure if any patterns we identify are 'accidental' or real. When a decision table is being created during migrating a legacy application, if we do not understand the purpose and background, we may interpret random correlations as genuine patterns and allow them to influence the table design.

Next, we can observe that the decision table has a specific pattern with alternating blanks in the second, third, fourth and fifth columns. As above the presence of these patterns hint some compression should be possible (assuming they reflect a genuine pattern in the business logic of course).

We should also observe that we are making decisions about cases with three input attributes, a **Type** which might be *numeric* or *categorical*, and an **Adjustment** and a **Loss** which are certainly *numeric*. The output attribute of the decision table is a **Classification**, which is *categorical* with three possible values. It will be assumed for the moment that **Type** is *categorical* and that there are only the seven mentioned types.

Now let us dig in a bit more. The decision table has 15 rows – so we have 15 rules. A quick examination shows that while the first rule always fires, the other 14 are mutually

exclusive. Essentially the first rule gives the ‘default’ **Classification**, and exactly one of the other fires if this is an ‘exception’ case.

Looking at the 14 rules we see that all the rules which produce a TOP outcome have the same structure, and all the ones which have a BOTTOM outcome have the same structure (giving rise to the patterns we noted above). We come back to this point later.

Another thing which strikes the eye is that in the columns relating to **Adjustment**, there are only two distinct values, 200 & 500. This again hints simplification ought to be possible. There is a greater variety of values in **Loss** columns which threatens to make things more challenging though.

3 A Minimal Solution?

If we look at the original example, the trick to making simplification is to sort the rows differently. When we sort the rows by card type and outcome it is then obvious that for a given card type, there is effectively a ‘default’ discount code, which applies to all cases except for a particular age range, so effectively the logic is to select on the card type, assign the default, then check for a special case. It is less obvious if this kind of simplification is possible in the target decision table, though there are some things which are suggestive.

Before we go any further, we might reasonably ask how well can we do? Can we work out some lower bound on the minimum number of rules (i.e. decision table rows) we need? Thinking about it we can. If there is a distinct value in a column, whatever solution we come up with must have a distinct rule/row that uses that value. There are four distinct values in the first **Loss** column, (all potentially leading to a classification of TOP) so we need four rules for those. Additionally, there are another four distinct values in the second **Loss** column, (all potentially leading to a classification of BOTTOM) so we also need four (different) rows for those. Looking in the **Adjustment** columns, we only get two distinct values. The value 200 is paired with two values of **Loss** which only appear in the rules for type 31. Since we have concluded we need distinct rules for the distinct **Loss** values which are paired with these, we see we do not necessarily require additional rules for this value of **Adjustment**. Likewise, in all the other scenarios the value of **Adjustment** is 500, so we should not need new rules for this value either[1]. Adding in the default rule, we can see the best we can hope to do is to get a decision table which has nine rules/rows (eight for the eight distinct values of **Loss** and then the default rule).

[1] If this sounds a bit convoluted, hopefully it becomes clearer when we see a ‘solution’

Let us start with our original table:

	Type	Adjustment	Adjustment	Loss	Loss	Classification
1						NONE
2	31	> 200		< -150		TOP
3	31		< 200		>= -189	BOTTOM
4	32	> 500		< -1000		TOP
5	32		< 500		>= -99	BOTTOM
6	33	> 500		< -1000		TOP
7	33		< 500		>= -100	BOTTOM
8	34	> 500		< -1000		TOP
9	34		< 500		>= -100	BOTTOM
10	35	> 500		< -800		TOP
11	35		< 500		>= -100	BOTTOM
12	36	> 500		< -800		TOP
13	36		< 500		>= -100	BOTTOM
14	37	> 500		< -2000		TOP
15	37		< 500		>= 0	BOTTOM

From the above analysis, it seems like **Loss** is the most significant attribute which limits our chances of simplifying matters, so a good move seems to be to try sorting the rows on **Loss**, to see what it looks like. This gives:

	Type	Adjustment	Adjustment	Loss	Loss	Classification
1						NONE
2	31	> 200		< -150		TOP
3	32	> 500		< -1000		TOP
4	33	> 500		< -1000		TOP
5	34	> 500		< -1000		TOP
6	35	> 500		< -800		TOP
7	36	> 500		< -800		TOP
8	37	> 500		< -2000		TOP
9	31		< 200		>= -189	BOTTOM
10	32		< 500		>= -99	BOTTOM
11	33		< 500		>= -100	BOTTOM
12	34		< 500		>= -100	BOTTOM
13	35		< 500		>= -100	BOTTOM
14	36		< 500		>= -100	BOTTOM
15	37		< 500		>= 0	BOTTOM

The effect of this is to highlight potential savings. We see the rows 2, 3 and 4 are identical for **Types** 32 to 34, rows 7 and 8 are identical for **Types** 35 and 36 and rows 10 to 13 are identical for **Types** 33 to 36. So, we can simplify as follows:

	Type	Adjustment	Adjustment	Loss	Loss	Classification
1						NONE
2	31	> 200		< -150		TOP
3	32,33,34	> 500		< -1000		TOP
4	35,36	> 500		< -800		TOP
5	37	> 500		< -2000		TOP
6	31		< 200		>= -189	BOTTOM
7	32		< 500		>= -99	BOTTOM
8	33,34,35,36		< 500		>= -100	BOTTOM
9	37		< 500		>= 0	BOTTOM

We now have a decision table which has nine rows, the number of rows which we earlier logically deduced is the minimum possible.

So, we are all done! Or are we? I think not.

4 Alternative Approaches

While the decision table above provides a sort of compression of the original, it feels to me that it is by cheating a bit, replacing a simple attribute test of the form:- '*is **Type** equal to X*', to a more complex membership test of the form:- '*is **Type** a member of the list [X_1, X_2, …]*' We have a smaller table, but we might ask ourselves, does this make it easier to maintain? For example, if the value for **Loss** in column 4, row 5 changes to -1000, should we now fold this case for **Type** 37 into the row 2? If the threshold in column 5 for **Type** 34 changes from -100 to -101, we will need to introduce a new row.

Do we have any other options? When I originally started thinking about the problem, I did consider looking at some machine learning options. Jacob Feldman's solution[2] beat me to this examining the approach in some detail using the **Rule Learner** tool. One should look at Jacob's submission to get full flavour of this, but my principal takeaway from his findings and also from the solution provided by Jack Jansonius[3] is that compressing this particular decision table is actually quite hard.

One could say it was a bad example to choose for the challenge. Or perhaps the fact it is hard makes it a good one.

As an alternative approach to compression than those offered by the decision tree methods explored by Jacob, I have come across the idea of Rough Sets[4]. As an approach this is directly applicable to the idea of building decision tables from data and significantly offers a direct way of establishing which rows and columns are 'relevant' for a decision. However, I barely scratched the surface of this approach yet and certainly am not sure how one might apply it to this particular example.

---

[2] See https://openrules.wordpress.com/2020/10/05/compressing-decision-tables/

[3] See https://dmcommunity.files.wordpress.com/2020/10/challenge2020sep.jackjansonius.pdf

[4] See https://en.wikipedia.org/wiki/Rough_set for the basics of Rough Sets

However, one could argue that using machine learning in this context is the wrong way to go about things. We are not trying to *learn* the rules, we already *know* them, it is just we think we have more than we need.

This suggests a different way to address the problem. Take a step back and ask, "*What is our business objective*?" or more specifically "*What is this decision table for, and if we want to compress it what kind of compression are we really looking for*?" The idea is to stop just considering the decision table as a fragment of code which we want to optimise but instead to think about what role it plays in our business problem, why we produced it in the first place and how do we think it might evolve.

This approach runs up against the major issue identified at the start – we have no context for the decision table. But if we make a few plausible assumptions I think we can come up with a superior solution than the one above, and one which gives us a decision system which is not only compressed, but (in my view) easier to maintain[5].

So, what assumptions should we make? From an examination of the decision table, these seem plausible:

- The decision table will change over time. The **Adjustment** and the **Loss** values will periodically be updated. New **Types** may be introduced, old ones retired.
    - If this is not the case, is it worth the effort of compressing it? And if it is why not optimise aggressively, hard coding it rather than with a decision system?
- The real thing the business is interested in is classifying a case based on the **Type** of the case. The **Adjustment** and the **Loss** are just the parameters used to make the classification.
    - This assumption echoes the original example on card types. What was important was classifying the discount code for Standard, Gold and Platinum cards. The applicable age ranges are the things which are likely to be modified, not the card type or discount codes.
- The classifications of a case, NONE, TOP and BOTTOM is not going to change.
    - To move beyond three classifications the neat pattern of the table as it stands would fall apart.
- To get a TOP classification, a particular case must have an **Adjustment** greater than some threshold, and a **Loss** less than some threshold, where the thresholds are determined by the **Type** of the case.
    - This assumption is that regardless of the actual values in the columns for **Adjustment** and **Loss** the logic of how we use them remains the same
- To get a BOTTOM classification, a particular case must have an **Adjustment** less than some threshold, and a **Loss** not less than some threshold, where the thresholds are determined by the **Type** of the case.
    - Again, the assumption is that the logic is stable

[5] Of course, this requires that the assumptions are correct!

The last two assumptions give us a route to simplification[6]. As the table stands, the *logic* for making a classification is the same for any case. It only depends on the actual thresholds which in turn depend on the **Type**. This suggests that we can make a separation between the *logic* of the classification, and the selection of the *parameters* which drive this logic.

So, we can come up with a decision table with one input column and four output columns to select our parameters[7]:

	Type	Upper Limit Adjustment	Lower Limit Adjustment	Lower Limit Loss	Upper Limit Loss
1	31	200	200	-150	-189
2	32	500	500	-1000	-99
3	33	500	500	-1000	-100
4	34	500	500	-1000	-100
5	35	500	500	-800	-100
6	36	500	500	-800	-100
7	37	500	500	-2000	0

We must the pass the parameters into the classification logic. If our decision technology allows us to create parameterised decision components, (which many tools do), we could do something like this to directly invoke the classification logic (which we assume to be encapsulated in a parameterised 'rule set' called 'classify'):

	Type	Classification
1	31	classify(200, 200, -150, -189)
2	32	classify(500, 500, -1000, -99)
3	33	classify(500, 500, -1000, -100)
4	34	classify(500, 500, -1000, -100)
5	35	classify(500, 500, -800, -100)
6	36	classify(500, 500, -800, -100)
7	37	classify(500, 500, -2000, 0)

We then need to separately define our *logic* as a set of (sequentially executed) rules. The following pseudo code outlines the general idea of how 'classify' would look:

[6] From what is in the table, we might also assume that the **Adjustment** threshold for TOP rules and for BOTTOM rules are always equal for the same **Type**. The solution presented allows them to differ.

[7] Note that if we drop the assumption that there are only seven types, we need to add one more row at the bottom with suitable parameter values which classifies any **Type** not listed as NONE (and make the table single hit)

```
rules classify(case, upperAdjustment, lowerAdjustment, lowerLoss, upperLoss)
{
      rule topRule:
            if case.adjustment > upperAdjustment
            and case.loss < lowerLoss
            then return TOP;
      rule bottomRule:
            if case.adjustment < lowerAdjustment
            and case.loss >= upperLoss
            then return BOTTOM;
      rule noneRule:
            if true
            then return NONE;
}
```

At this point we have 10 rules, 7 in our decision table and 3 in our ruleset. Not quite the 'maximal' compression of the solution in the previous section. But we do get several benefits in terms of maintainability.

- The solution has a stable structure. No matter what changes in terms of the parameters in the **Adjustment** or **Loss** parameters we always have 10 rules. The number does not go up (or down) as we make changes. The number of rules now depends on how many **Types** we have (which we are assuming is the key business concept), not how many distinct parameter values we have.
- If we add a new **Type**, we only add one row (not two as would be necessary in the original formulation) so we have an intrinsic 50% compression.
- If we want (and are prepared to forgo the first benefit), we can still use the compression method of the first solution where the parameters are identical for more than one **Type**. So, we could combine the rows for 33 and 34 and for 35 and 36 to end up with only eight rules – less than the theoretical minimum we deduced based on building a solution with only one decision table.
- Perhaps most importantly the business logic is more explicit, we see precisely how **Adjustment** and **Loss** are used so we better understand what we are doing!

5 Conclusions

A few closing thoughts on the challenge:

- Understand what a decision table is for, before thinking about compressing it! There is a trade-off between compression and maintainability. Before trying to compress a decision table, we need to think about how it is likely to evolve over time. If we do not, we may find compressing it creates more problems than it solves.
- Some decision tables are easier to compress than others. The table we are asked to compress offers rather little in the way of opportunities.
- We can get a handle on the limits of compressibility by looking at how many distinct values of the table attributes there are. Fewer allow for more compression.
- We should consider divide and conquer approaches, such as separating logic from parameter selection, as a compression strategy.
- I need to look more deeply into how supervised machine learning techniques might be applied to this kind of problem.