Temporal Rules and Temporal Decision Trees: A C4.5 Approach

Kamran Karimi and Howard J. Hamilton
Technical Report CS-2001-02
December, 2001

Department of Computer Science
University of Regina
Regina, Saskatchewan
CANDA S4S 0A2

ISSN 0828-3494
ISBN 0-7731-0429-1

Temporal Rules and Temporal Decision Trees: A C4.5 Approach

Kamran Karimi and Howard J. Hamilton
Department of Computer Science
University of Regina
Regina, Saskatchewan
Canada S4S 0A2
{karimi, hamilton}@cs.uregina.ca

Abstract - Decision trees are useful tools for classification and prediction purposes, and have been applied mostly to data that is void of any explicit notion of time. This covers many application areas where the data is about populations of the same entities, but is not suitable for cases where there is a temporal relation among the data. One case is when we are gathering data about a single system over time. In this paper we use a toy domain to show how flattening such data by merging consecutive records into a single one enables a classifier such as C4.5 to be used for rule discovery over time. C4.5 does not understand time, so all temporal information is lost in the output. In this paper we propose to include time in decision trees and rules derived from them. We have modified parts of the C4.5 Release 8 package to make time explicit and generate temporal rules and decision trees. We show that building a temporal decision tree is an instance of solving a Constraint Satisfaction Problem. We finally show how temporal rules or temporal decision trees can be interpreted as plans for robots with sensors and actuators. Such plans tell us what to do at each time step, and what to expect in the following time step.

1. Introduction

A decision tree uses a set of condition attributes to determine the value of a decision attribute by performing a series of tests. Many examples of condition and decision attributes are saved in data *records* and fed to the decision tree generator. Decision trees have been used in different applications like classification and rule generation. In most cases, they have been used to process data generated from *populations* of entities like patients, cars, flowers, and so on [3, 17]. In such cases there is no temporal order among the data records. Census data for example, is gathered at nearly the same time from different people. However, sometimes we want to study a single entity over time. In this case the data come from a single source, and we want to discover trends and rules that exist over time.

One interesting example is a robot that interacts with its environment through a series of sensors and actuators. To be able tomake plans, it needs to know how its current actions affect the environment in the future. The results of an action such as "move" will be known to it later. This kind of problem is encountered in any situation where there is a delay between performing an action and observing the results of that action. This makes the passage of time an important consideration.

In this paper we propose to make time explicit in both decision trees and the rules generated from them. Section 2 introduces a toy domain where there is a temporal order among the data. In Section 3 we show how a decision tree's output can be interpreted as a set of temporal rules without the need for any consideration of time in the process of building the tree. Section 4 proposes that we build a temporal decision tree right from the beginning and shows that this is a Constraint Satisfaction Problem [12]. Section 5 gives an example of the usefulness of a temporal rule or temporal decision tree as a plan. Section 6 concludes the paper.

2. Temporal Data

Temporal data appear in many application areas [16]. As an example, suppose a robot can sense variable x from the environment and set variable a to affect the environment. x could denote the position of the robot along a

single axis, and *a* could denote one of the two actions L and R, for moving to the left or to the right respectively. We observe new values for *a* and *x* and create a record <*a*, *x*> of their values at every time step. This grouping of events has been done in [13] too, where the phrase *Multiple Streams of Data* is used to describe simultaneous observations. In other applications such records could be considered to form *time series* [4]. In our case we can assume that *a* is determined randomly, while *x* is determined from the environment. The robot can have a training phase during which such data is collected.

Not much can be discovered by using these records, because discovering the effects of the current action on the current position requires having access to the records that are generated later. To provide all the relevant information together, we *flatten* the data by placing consecutive records in a single record. The number of records that are merged is called the *time window size*. We also rename the attributes in the flattened record so as to avoid name clashes. Figure 1(a) gives an example of the original records, where time moves from top to bottom. In Figure 1(b) we have flattened the records in Figure 1(a) using a window size of 2. In the flattened records, time passes from left to right as well as from top to bottom.

Time Step	<*a*, *x*>
1	<L, 1>
2	<R, 0>
3	<R, 1>
4	<R, 2>

a

Time Step	<a_1, x_1, a_2, x_2>
1, 2	<L, 1, R, 0>
2, 3	<R, 0, R, 1>
3, 4	<R, 1, R, 2>

b

Figure 1. Records of the agent's position and action

Notice that with the exception of the first and last records, each record appears twice in the flattened records: once as the second part (the effect), and then as the first part (the cause). The window size depends on the domain. Assuming time passes in discrete steps, if an interaction among the attributes takes *t* time steps to show its results, then we should use a window size of at least *t*. A domain expert should determine the appropriate value of *t*.

We will use C4.5 [15] as our decision tree and rule generator. It considers the last value in a record to be the decision attribute (or the class, as it is called by C4.5). In the examples in this paper we are interested in predicting the next value of *x*, and that is why it comes last. Feeding the flattened records to C4.5, we get rules of the form [7]: **if** {(x_1 = 0) AND ((a_1 = R)} **then** (x_2 = 1).

Karimi and Hamilton [7] have used URAL [18], a simulator for a more complex Artificial Life [9] domain, to generate test data, and employed C4.5 to extract the kind of rules mentioned in the previous paragraph. There it was shown that C4.5's ability to prune the irrelevant attributes and come up with rules that made sense were good. However, its generated rules did not reflect the temporal orderings of the domain. For example it had to be kept in mind that x_2 represented the same thing as x_1, but at the next time step.

3. Temporal Rules

Temporal rules have usually been discovered by association mining [11, 13], but the exact timing of events is often lost in the output. In the previous section, time completely disappeared from the output, because we concealed the passage of time among the observed records by flattening them and thus lost temporal information. This allowed a decision tree builder like C4.5 to discover rules over time, but there was no explicit notion of time in the results.

We can reverse the effects of flattening by bringing back the concept of time in the generated rules. For example, we can interpret the rule in Section 2 as saying: **if** {At Time 1: (*x* = 0) AND (*a* = R)} **then** At Time 2: (*x* = 1). Notice how we changed the attribute names back to their original form and added an *At Time* phrase when we were about to encounter a attribute at another time step. Time in each flattened record is measured relative to the start of the record.

A decision tree generator like C4.5 assumes that all the condition attributes are available at the same time. The resulting decision tree can use these condition attributes in any order, and so might change their temporal order to make shorter trees. As we just saw, we can use C4.5 to generate rules from flattened records and then modify the rules to bring back temporal order. Suppose we have the rule **if** {(a_3 = 3) AND (a_1 = 2) AND (a_2 = 6)} **then** (b = 8) that comes from flattened records. In general, we perform the following steps to make it a temporal rule:

1. Reorder the arguments in the rule, so that the attributes appear in the same temporal order as their respective records. For example, if in the flattened record a_3 appears in the second record, while a_1 and a_2 appear in the first record, then the rule becomes:
   if {(a_1 = 2) AND (a_2 = 6) AND (a_3 = 3)} **then** (b = 8).
2. Add "At Time n:" before the attributes that happen at time step n in the flattened record. If the last condition attribute and the decision attribute occur at different time steps, we add an "At Time n:" before the decision attribute. Intuitively, this can be considered the reverse of the flattening operation. The rule now becomes:
   if {At Time 1: (a_1 = 2) AND (a_2 = 6) AND At Time 2: (a_3 = 3)} **then** At Time 3: (b = 8).
3. Now change the attribute names back to their original forms before they were flattened. This might convert our example rule to:
   if {At Time 1: (a_1 = 2) AND (a_2 = 6) AND At Time 2: (a_1 = 3)} **then** At Time 3: (a_2 = 8).

Depending on the domain, some may opt to interpret such temporal rules as *causal relations*, even though the term seems to over used [5]. In our toy domain for example, we can call the current x position and the current action as the causes for the next x position. In a domain with causal relations, the time window size is the maximum amount of time during which we suspect an event can cause effects.

We have modified *c4.5rules*, a program in the C4.5 Release 8 package that generates rules from decision trees, to output temporal rules. It can now handle flattened records as input. The user can specify the number of records involved in the flattening process via a new option -T (Time window). for example, using '-T 2' would be appropriate for the temporal data of Section 2. The c4.5rules program then generates the rules as usual, but before outputting them, it sorts the decision attributes of each rule according to their time of occurrence. It then prints out the rules, along with the temporal information as outlined in the example rules given above. Using the new c4.5rules program does not require modifying the rest of the C4.5 package.

There is no need to rename the attributes of the flattened records before feeding them to the modified c4.5rules program. C4.5 does not care about attribute names (it differentiates among the decision attributes by using their position in the input), and the temporal information at the output disambiguates the results, so there will not be any name clashes in the temporal rules.

4. Real-Time Decision Making

In the previous section we changed the rules generated from a *normal* decision tree so that it included information about the passage of time. Another approach to the problem is to build a temporal decision tree in the first place. In a temporal decision tree the decision attributes are encountered in their correct temporal order as the tree is traversed from the root down.

In a normal decision tree, the condition attributes may be used in a different order than the one determined by the input files. This is done to make the tree shorter, but in a temporal domain makes it impossible to traverse the tree as data are gathered, as in general we have to wait for all records to be available. In the previous section for example, we did not have the value of a_3 before a_1. This point is illustrated in Figure 2 for a window size of 3.

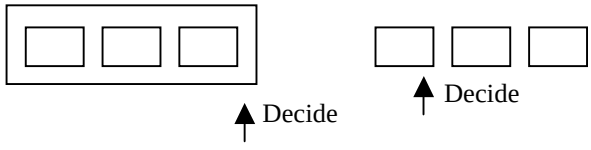

Figure.2. At left, a normal decision tree has to wait for all the records in the window. At right, a temporal tree uses the records for decision making as soon as it encounters them.

Here each box represents an unflattened record, and time passes from left to right. On the left we see the situation in ordinary decision trees, where we have to wait for up to three records before we can start traversing the tree because we have to go back and forth between the attributes in the 3 records while doing so. On the right, we use each record as we see it. We are sure that once we are past a record (whether we used the attribute in it or not), we will never go back to it. In other words, like temporal rules, time never goes back in a temporal decision tree.

Like a temporal rule, a temporal decision tree allows for real-time decision making by ensuring that we use the attributes in the tree in their original temporal order. Normal decision tree builders rank the condition attributes according to how suitable they will be for expanding the tree at each step. For a temporal decision tree, the attributes should be ranked according to their temporal order as well as their suitability for expanding the tree. The condition attributes thus have to be partitioned according to their time of encounter. We can partition the condition variables according to their temporal order. So attributes that are encountered in time step i of the flattened records go into set T_i. In our toy domain, for example, we have two sets $T_1 = \{x_1, a_1\}$ and $T_2 = \{x_2, a_2\}$ corresponding to the two time steps. If at a node of the tree a condition attribute from the set T_i is used, then the children of that node can only use condition attributes from the sets T_j ($j \geq i$), even if doing so makes the tree sub-optimal. An example temporal decision tree is shown in the right part of Figure 3.

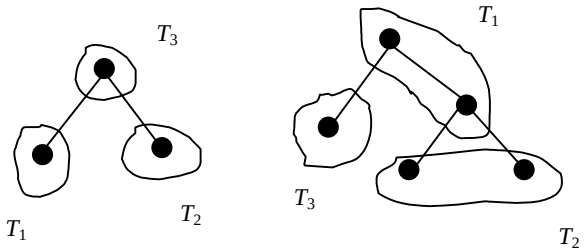

Figure 3. A normal decision tree at left and a temporal decision tree at right. Only decision attributes are shown.

Building a temporal decision tree is a Constraint Satisfaction Problem, where the choice of one attribute limits the future choices in that branch of the tree. The greedy method used by C4.5 will not work properly because choosing a condition attribute at any time step prevents us from using attributes in previous time steps. This means that some choices of the normal C4.5 algorithm could result in a very bad decision tree, because we may run out of available condition attributes. As is normal in constraint satisfaction problems, this would require us to backtrack up the tree branches and choose other condition attributes.

This new optimization problem is combinatorial in nature, and one can use heuristics to limit the size of the search space if needed. For example, since we know the tree that our unmodified algorithm can produce (the one built with no regard for temporal orderings), we can stop the search in the space of temporal trees as soon as we find a tree that is "close enough" to this tree. Another heuristic is to search for a maximum amount of time, and return the best tree found within that time limit.

We have modified the c4.5 program (C4.5's decision tree builder) to consider the temporal order while building the tree. As in the c4.5rules program, a new option -T is added for the user to specify the time window size. It still ranks the decision attributes as usual, but once an attribute from a certain time step is used, it disregards all attributes from previous time steps. Our implementation does not perform backtracking. The test results have been as expected: trees either do not change, because temporal order is already satisfied, or they get smaller while the error rate increases, because C4.5 now refrains from using some of the decision attributes. To see the effects of the time window of the generated tree, we interpreted the data in a Letter Recognition database [2] as being temporal. The data consists of 20,000 records to classify the letters of the English alphabet. There are 16 condition attributes that are actually seen all at the same time. However, we could look at the data as if different parts were generated at consecutive time steps. The results are shown in figure 4, where a time window of 1 results in C4.5's normal behavior because all attributes are assumed to be produced at the same time.

Time Window	Before Pruning		After Pruning	
	Size	Error	Size	Error
1	26721	0%	25713	0.5%
3	6689	32.9%	6385	33.1%
4	6689	32.9%	6835	33.1%
5	6689	32.9%	6385	33.1%
6	6081	33.8%	5793	34.0%
7	6081	33.8%	5793	34.0%
8	6225	33.4%	5937	33.5%
10	6225	33.4%	5937	33.5%
12	6225	33.4%	5937	33.5%
14	6225	33.4%	5937	33.5%
15	6225	33.4%	5937	33.5%
16	1377	59.4%	1345	59.4%

Figure 4. The effects of the time window on the size and accuracy of the tree.

In all the cases the data remain the same, and only their interpretation changes. In other words, in each case we consider different condition attributes to have happened at the same time. For example, with a time window of 4, each 4 neighboring attributes are supposed to belong to the same time step. With a time window of 5, there are 3 condition attributes at each time step, with the last time step containing only 1 condition attribute. The size and the quality of the tree change as the time window changes, because the condition attributes will move from one time step to another. Increasing the time window beyond 16 (the number of condition attributes) has no effect on the results. There is a small increase in the quality of the tree when going from a time window of 5 to 6. This is because the rearranging of the attributes that results from the consideration of time has helped the algorithm.

The ordering of the nodes is the main difference between a normal and a temporal decision tree. It is also possible to make the time step of each node in the tree explicit by retaining the index i of the set T_i where the attribute corresponding to that node was chosen from. Doing this makes it unnecessary to rename the attributes, as name clashes are prevented through the use of the time step values. Temporal rules can readily be generated from such decision trees, because the reordering of attributes has already been done.

After the tree is built, we can use it to process data. With a window size of t, we need t different instances of the decision tree working in parallel. The same record is considered to be at different time steps in each decision tree instance. This is depicted in Figure 5 for the case of a window of size 3. Here time moves from left to right.

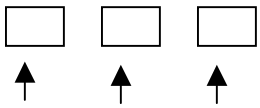

Figure 5. Three consecutive records being used by three decision trees.

5. Temporal Rules and Decision Trees as Plans

Temporal sequences are often considered to be passive indicators for the presence of temporal structure in data [1, 6, 11, 13], but when there are causal relations in the domain, a temporal decision tree or temporal rule can be interpreted as a plan. As an example, if the data records are produced by a robot with a set of actuators, A, and a set of sensors, S, then some of the decision attributes in a decision tree are under the robot's control, while others are not. Suppose the leaves of the tree represent a attribute from S. In the example given in Section 2, we have $A = \{a\}$ and $S = \{x\}$. In such cases, making time explicit in a temporal rule or a temporal decision tree allows them to be interpreted as plans that can be followed over time to reach a goal.

We will show that a temporal rule generated automatically from a robot's observations of its environment can be considered a plan that allows the re-generation of the same results. The idea of a rule being an executable entity is not new. It appears in [10] for example, where rules are considered logic programs in a situation calculus domain. As in [8], in many cases a robot's plans are written by domain experts. In what follows we make time completely explicit in a plan, and make a clear distinction between what an agent can do through its actuators (input to a logic program) and what it can expect in return (the logic program's output) at each time step. Here the plans are generated automatically from the robot's previous observations. Notice how a logic program and a robot's perspectives are reversed, as a robot's outputs to its actuators becomes a logic program's input, and the same reversal happens for the robot's sensors. This is because a logic program executes the rules, which gives it the role of the environment.

We consider the robot's goal to be changing its environment. In other words, it wants to set a attribute from *S* to a specific value. To do this the agent can choose appropriate rules, or branches of the tree that lead to the desired result, and then use the attributes from set *A* at each time step to get closer to the goal.

Attributes from *S* are regarded as *execution context* for attributes from *A*. At each time step, the robot can verify if the plan is succeeding or not by comparing the actual context attributes from its sensors against their values as predicted by the decision tree or rule. If they match, it applies the proper values to its actuators as needed, and goes to the following time step. If the observations do not match the expectations, then the robot could try to find another rule that leads to the goal and continue from there. The plan fails if no such rule can be found. In the example in Section 2, if the robot is currently at ($x = 0$) and the goal is to go to ($x = 1$), it applies the action ($a \leftarrow$ R). If this does not result in a change in the robot's position (maybe the wheels are stuck) then it could try ($a \leftarrow$ R) again.

In situation calculus we have the same type of plans: We perform an action in a situation (context) and move to the next situation [14]. But with a pure situation calculus approach we may have difficulty finding out which attributes are the important ones in a transition (this information should usually be encoded in the plan by a domain expert). C4.5 with temporal modifications can be used as a tool to identify relevant condition attributes and generate plans from observations [7]. In general, we represent a temporal rule as: $C_{sn+1} \Leftarrow t_1, t_2, \ldots t_n$ where each of the t_i is defined as: $t_i \equiv C_{si}, C_{ai}$. Here n is the time window used in the flattening process.

C_{si} represents a set of readings of relevant sensor attributes at time step i. The attributes in C_{si} are a subset of the attributes in *S*. One example from Section 2 is $C_{s1} = \{(x = 1)\}$. C_{ai} is a set of value assignments for the actuators at time step *i*. Like C_{si}, the attributes in C_{ai} are a subset of the attributes in *A*. For the data in Section 2 we could have $C_{ai} = \{(a \leftarrow \text{L})\}$. Each of the C_{si} and C_{ai} can be empty, which means that nothing will be checked, or done, respectively, at time step *i*. The *relevancy* of the attributes in each t_i could be determined automatically by C4.5.

While following a rule, at each time step *i*, the conditions in C_{si} should be checked first. If they all hold, the assignments in C_{ai} are applied, and the robot moves to the next time step. Since time is measured relative to the start of the rule, the robot may be following more than one rule at the same time and it may be at a different time step in each rule.

Sometimes there is no causality relation among the consecutive time steps. In other words, following a rule up to t_i ($i < n$) does not guarantee that C_{si+1} will be valid. In such cases t_i is read as: *in case* the conditions in C_{si} hold, do C_{ai}. In other times there is a causal relation among the consecutive time steps, and after step *i* one can expect to see C_{si+1} to hold before applying C_{ai+1}. Here t_i is read as: C_{si} *is expected* to hold, and if this is indeed true, then do C_{ai}.

6. Concluding Remarks

Decision trees have traditionally been generated with no regard for time. This reflects the common form of data used in many applications, but does not include domains with a need for explicit representation of time. Flattening makes time implicit and allows a program like C4.5 to build decision trees and extract rules from data that span time. Flattening can be performed as a preprocessing phase, and enables us to use tools that do not consider the passage of time. The negative side effect is that the notion of time disappears in the results. In this paper we

proposed that we complement the flattening process by making time explicit in both the decision trees and the rules generated from them. As we have shown, temporal rules can easily be generated from ordinary decision trees by reordering and, if needed, renaming of the attributes. Nothing needs to be changed in the tree-building algorithm. Building a temporal decision tree, on the other hand, is a Constraint Satisfaction Problem. Such decision trees can be regarded as plans if the rules involve a set of observed attributes and a set of actions under the agent's control that can affect the observed attributes.

The modified c4.5rules and c4.5 programs retain backward compatibility, and their output is unchanged when the new option is not used. The modifications are available in the form of a patch file that can be applied to standard C4.5 Release 8 source files. It can be downloaded by contacting the authors or from http://www.cs.uregina.ca/~karimi/downloads.html

References

[1] Agrawal R. and Srikant R., Mining Sequential Patterns, *Proceedings of the International Conference on Data Engineering (ICDE)*, Taipei, Taiwan, March 1995.

[2] Blake, C.L and Merz, C.J., UCI Repository of machine learning databases http://www.ics.uci.edu/~mlearn/MLRepository.html. Irvine, CA: University of California, Department of Information and Computer Science, 1998.

[3] Bowes, J., Neufeld, E., Greer, J. E. and Cooke, J. A., Comparison of Association Rule Discovery and Bayesian Network Causal Inference Algorithms to Discover Relationships in Discrete Data, *Proceedings of the Thirteenth Canadian Artificial Intelligence Conference (AI'2000)*, Montreal, Canada, 2000.

[4] Berndt, D.J. and Clifford, J., Finding Patterns in Time Series: A Dynamic Programming Approach, *Advances in Knowledge Discovery and Data Mining*. U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, et al. (eds.), AAAI Press/ MIT Press, pp. 229-248, 1996.

[5] Freedman, D., and Humphreys, P., *Are there Algorithms that Discover Causal Structure?*, Technical Report 514, Department of Statistics, University of California at Berkeley, 1998.

[6] Guralnik, V., Wijesekera, D. and Srivastava, J., Pattern Directed Mining of Sequence Data, *Proceedings of the Fourth International Conference on Knowledge Discovery & Data Mining (KDD-98)*, 1998.

[7] Karimi, K., and Hamilton, H. J., Finding Temporal Relations: Causal Bayesian Networks vs. C4.5, *The 12th International Symposium on Methodologies for Intelligent Systems (ISMIS'2000)*, Charlotte, North Carolina, October 2000.

[8] Levesque, H. J., Reiter, R., Lespérance, Y., Lin, F., and Scherl. R., GOLOG: A Logic Programming Language for Dynamic Domains, *Journal of Logic Programming*, 31, pp. 59-84, 1997.

[9] Levy, S., *Artificial Life: A Quest for a New Creation*, Pantheon Books, 1992.

[10] Lin F., and Reiter, R., Rules as Actions: A Situation Calculus Semantics for Logic Programs, *Journal of Logic Programming Special Issue on Reasoning about Action and Change*, 31(1-3), pp. 299-330, 1997.

[11] Mannila, H., Toivonen, H. and Verkamo, A. I., Discovering Frequent Episodes in Sequences, *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, pp. 210-215, 1995.

[12] Nadel, B. A., Constraint Satisfaction Algorithms, *Computational Intelligence*, No. 5, 1989.

[13] Oates, T. and Cohen, P. R., Searching for Structure in Multiple Streams of Data, *Proceedings of the Thirteenth International Conference on Machine Learning*, pp. 346 - 354, 1996.

[14] Poole, D., Decision Theory, the Situation Calculus, and Conditional Plans, *Linköping Electronic Articles in Computer and Information Science, Vol. 3 (1998): nr 3*, http://www.ep.liu.se/ea/cis/1998/008, June 15, 1998.

[15] Quinlan, J. R., *C4.5: Programs for Machine Learning*, Morgan Kaufmann, 1993.

[16] Roddick, J.F. and Spiliopoulou, M., *Temporal Data Mining: Survey and Issues*, Research Report ACRC-99-007. School of Computer and Information Science, University of South Australia, 1999.

[17] Silverstein, C., Brin, S., Motwani, R. and Ullman, J., Scalable Techniques for Mining Causal Structures, *Proceedings of the 24th VLDB Conference*, pp. 594-605, New York, USA, 1998.

[18] http://www.cs.uregina.ca/~karimi/downloads.html/URAL.java