Classification Based on the Trace of Variables over Time

Frank Höppner and Alexander Topp

University of Applied Sciences Braunschweig/Wolfenbüttel
Robert Koch Platz 10-14
D-38440 Wolfsburg, Germany

Abstract. To be successful with certain classification problems or knowledge discovery tasks it is not sufficient to look at the available variables at a single point in time, but their development has to be traced over a period of time. It is shown that patterns and sequences of labeled intervals represent a particularly well suited data format for this purpose. An extension of existing classifiers is proposed that enables them to handle this kind of sequential data. Compared to earlier approaches the expressiveness of the pattern language (using Allen et al.'s interval relationships) is increased, which allows the discovery of many temporal patterns common to real-world applications.

1 Introduction

In knowledge discovery, in particular in classification tasks, the values of some variables are used to predict the value of another variable, the so-called class variable or label. Classifiers are trained by presenting historical cases including the class value for training purposes. Each historical case usually represents a real world object at some specific point in time (usually the time of recording the case into the database). If the variables are volatile, however, then the class label may depend on the *history* or *development* of the variables rather than their value at the time of recording only. If measurements are numerical and taken periodically we obtain time series, which we may analyze directly or feed into a feature extraction process (such as the extraction of Fourier coefficients) and use these features in traditional classifiers. Problems arise, however, if the values are not measured at a constant rate, if there are gaps in the data or we have mixed data types (not only numerical). As an example from health care, we may measure a numerical ECG signal together with external influences such as medication.

Rather than using information from a single point in time we are interested in predicting the class by analyzing the complete history of the variables (such as a patient's file or log files for technical systems). The remainder of the paper is organized as follows: In the next section, we will motivate the kind of data representation that we are going to use, namely sequences of labeled intervals. In section 3 we give a brief overview of related work on the analysis of interval

sequences in the literature and discusses drawbacks of the existing approaches. Section 4 presents a new approach to this problem that greatly improves the expressiveness of patterns compared to earlier proposals. In section 5 we present some results from fault diagnosis and finish with the conclusions in section 6.

2 Representing Data Histories by Interval Sequences

In some cases, the state of an object in the real world changes monotonically. For instance, a customer purchases more and more products from a company, but usually does not send it back to the retailer at some later point in time. Thus, the set of purchased items grows but an item once bought is never removed from this set. In such a monotonic setting, it is sufficient to store when another product has been purchased. Time-stamped event sequences can be used to represent this kind of data.

If we consider objects of the real world in general, however, most of them do not disclose such a monotonic behavior. If we observe the operation of an electrical device, we observe changes of its internal state from 0 (off) to 1 (on) but also from 1 to 0. In a medical domain symptoms may occur and vanish again later on. The demand for a certain product may be overwhelming today but negligible some months later.

One could still use event sequences (one event indicating that the symptom occurs and another when it vanishes) to represent these non-monotonic changes of the world. Suppose we have three devices A, B and C and that we observe some malfunction but cannot tell exactly when and why the failure occurs. We have recorded several histories and want to induce knowledge about the circumstances of failure from this data. Let us assume that the problem arises if devices are turned on in the order A, B, C but turned off in the order A, C, B. The goal is to *discover* this pattern in our database of historical cases. When still using event sequences to represent this data, the target sequence is represented by the following sequence:

$$\mathbf{A}_{\text{on}}\ \mathbf{B}_{\text{on}}\ \mathbf{C}_{\text{on}}\ \mathbf{A}_{\text{off}}\ \mathbf{C}_{\text{off}}\ \mathbf{B}_{\text{off}}$$

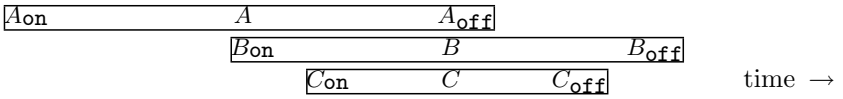
Since the seminal work in [1, 2] many algorithms have been proposed for the analysis of event sequences. Our target sequence will be discovered by any of these approaches if it is found frequently in the histories and helps to discriminate the two classes (fault/no fault). In all approaches the sequence above is found being a subsequence of the following:

$$\mathbf{A}_{\text{on}}\ \mathbf{B}_{\text{on}}\ A_{\text{off}}\ \mathbf{C}_{\text{on}}\ A_{\text{on}}\ \mathbf{A}_{\text{off}}\ \mathbf{C}_{\text{off}}\ \mathbf{B}_{\text{off}}$$

However, this case does **not** correspond at all to our target pattern: It is not true that the devices are turned on in the order A, B, C, because A has already been turned off before C is turned on. Only the fact that A is reactivated later makes it possible to match the event sequence here. Thus, our quite simple target problem cannot be represented and matched correctly. The reason is that events in an event sequence are considered independent from each other, but in fact every 'power-on' event corresponds to the very next 'power-off' event referencing

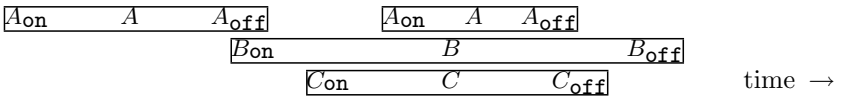
the same device. When expressing the temporal pattern as a sequence of events, we would have to impose such constraints explicitly (but this is usually not done in the literature on event sequences).

The dependency between corresponding events is better prevailed if both events form a single data object rather than two independent ones: one interval for each up-time (and/or down-time) of the device. Now, using intervals the pattern looks like this:



Next, the notion of being a subsequence must be defined for interval sequences. Firstly, for P being a subpattern of sequence S all intervals in P must occur in S (just like with events). Secondly, the desired relationships between the intervals has to be formulated, just as the (partial) order in case of event sequences. Allen et al. [3] has introduced 13 interval relationships that are shown in Fig. 1. The relationship between intervals is more complex than that of events, but the concept remains the same: For any two intervals A and B the set of allowed relationships has to be specified. In our case, we could require the following pairwise relationships: A overlaps B and C, B contains C.

If we now revisit our second historical case, it is represented by the following interval sequence:



When trying to match the intervals in this sequence to our pattern we have two possibilities for A: the first A correctly overlaps B but does not overlap C and the second neither overlaps B nor C. So this notion of '*being a subsequence*' corresponds much better to our intention and does not mistake this sequence as an occurrence of target pattern.

This example consisted of dichotomous variables only, but variables at other scales can also be used (cf. Fig. 2). For instance, a new interval may be introduced whenever a categorical variable changes its value or a numerical variable changes its qualitative behavior (e.g. increasing/decreasing, low/medium/high, etc., see also [4]).

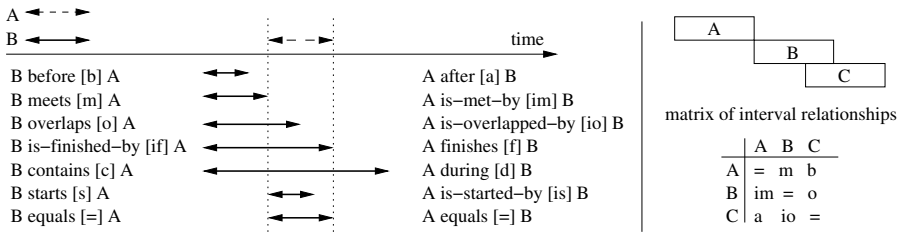


Fig. 1. There are 13 possible relationships between two intervals.

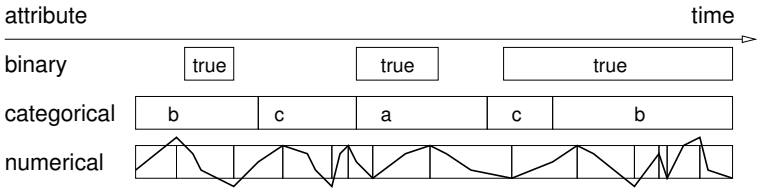


Fig. 2. Transformation of historical data into an interval sequence.

3 Related Work

More formally, we define an interval sequence $s = (s_i, l_i, e_i)_{i=1..n}$ as a finite sequence of intervals $[s_i, e_i] \in \mathbb{R}^2$ with labels $l_i \in L$, where $i = 1..n$, $s_i < e_i$ and L is a finite set of labels. Our data set consists of several such sequences, each carrying an additional class label for classification. We denote the set of 13 interval relationships by $\mathcal{I} = \{b, a, m, im, o, io, if, f, c, d, s, is, eq\}$ (cf. Fig. 1).

In the literature there is comparatively little work on the analysis of interval sequences for the purpose of classification or knowledge discovery. Most of them have their own notion of a *pattern* (subpattern of an interval sequence) and how it is matched against another sequence. We can compare the different notions of a pattern using the following unifying view. A pattern P is captured by the following matrix, where the m rows and columns denote intervals (resp. their label) $k_j \in L$ and the matrix elements $R_{i,j} \subseteq \mathcal{I}$ the set of possible pairwise relationships:

	k_1	k_2	...	k_m
k_1	$\{eq\}$	$R_{1,2}$	...	$R_{1,k}$
k_2	$R_{2,1}$	$\{eq\}$	...	$R_{2,k}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
k_m	$R_{1,2}$	$R_{1,3}$	...	$\{eq\}$

On the diagonal, where we compare identical (sets of) intervals k_j, the relationship has to be *equals* ($R_{j,j} = \{eq\}$). A pattern P matches a sequence $S = (s_i, l_i, e_i)_{i=1..n}$ of length n, if we can find an injective mapping $\varphi : \{1, 2, .., m\} \to \{1, 2, .., n\}$ such that (a) label k_j in the pattern equals label $l_{\varphi(j)}$ in the sequence and (b) the interval relationship r between $[s_{\varphi(i)}, e_{\varphi(i)}]$ and $[s_{\varphi(j)}, e_{\varphi(j)}]$ is contained in $R_{i,j}$ for all $i, j \in \{1, 2, .., m\}$.

If $R_{i,j} = \mathcal{I}$ holds for all $i \neq j$, any interval relationship would suffice and matching of patterns reduces to an existence test for all interval labels (just as in association rule mining). If we restrict $R_{i,j}$ to a subset of $\{a, b, eq\}$ we can simulate event sequences where two events either occur in parallel or in succession. Let us now consider how some approaches to the analysis of interval sequences fit in this view:

- In [5] the relation between k_1 and k_2 is always unique ($|R_{1,2}| = 1$). For any further interval k_j the relationship is again unique with respect to the smallest interval that encloses all preceding intervals $k_1...k_{j-1}$, but this leaves several possibilities open for the individual interval comparison $R_{i,j}$. As an

example, if $R_{1,2} = \{o\}$ and the union of k_1 and k_2 again *overlaps* k_3 we have $R_{1,3} = \{o, m, b\}$ and $R_{2,3} = \{o, s, d\}$.
- In [6] the relation $R_{i,j}$ is either *contains*/*during* or remains unspecified.
- In [4] all possible relationships $R_{i,j}$ are exploited, given that $|R_{i,j}| = 1$.
- Other authors prefer other relationships over Allen et al.'s, such as *is-older-than* or *survives* [7]. However, since the 13 relationships by Allen et al. are complete, those relationships can always be represented as a disjunctive combination of relationships in $\mathcal{I}$.

The approach in [4] is quite general in the sense that *all* $R_{i,j}$ are configurable whereas the other approaches focus either on a few $R_{i,j}$ or on a specific subset of $\mathcal{I}$ only. On the other hand, *all* pairwise relationships are strictly fixed in this approach, but many relationships that occur frequently in practice cannot be expressed by a single relationship $r \in \mathcal{I}$. For instance, '*A starts within B*' clarifies the location of the start time A_{on} of A with respect to B ($A_{\text{on}} \in]B_{\text{on}}, B_{\text{off}}[$), but the position of A_{off} is left open. Therefore, a number of different interval relationships between A and B are possible: *A is-overlapped-by B*, *A during B*, *A finishes B*. To reflect the true relationship correctly, $R_{i,j}$ should be chosen as $\{io, d, f\}$. Another example is '*A and B are disjoint*' that can be resembled by *after* and *before* relationships ($R_{i,j} = \{a, b\}$).

4 Increasing the Expressiveness

The goal of this work is to relax the conditions on the sets of interval relationships $R_{i,j}$ to increase the expressiveness of the pattern language and thus to discover more realistic dependencies. We want to achieve this in two steps:

Firstly, rather than specifying *all* relationships between any two intervals, we want to restrict ourselves to those relationships that are really helpful for the classification. By reducing the number of constraints in this way, the patterns become more robust against effects of dilation and translation and an undesirable rule fragmentation is avoided. As an example, suppose the true relationship is '*A and B occur before C*', which is quite a simple pattern. It does not contain any information about the relationship between A and B, so all 13 relationships are possible. The approach of [4] is capable of learning the relationship '*A before C*' and '*B before C*' but also requires a single relationship between A and B. As we have seen, there is no such single relationship, so if we are forced to fix this relationship artificially, the support of this pattern is fragmented into 13 rules. A rule evaluation measure would rank each of these 13 rules much lower than a rule that corresponds to the true pattern.

Secondly, we want to relax the requirement $|R_{i,j}| = 1$ such that relationships beyond the basic ones can be discovered (e.g., *does not intersect*: $R_{i,j} = \{$*after*, *before*$\}$).

4.1 Choice of the Classifier

We use a standard rule (or tree) learner that develops its model incrementally, such as ITRule [8] or C4.5 [9]. Such algorithms have already been used for

temporal data in the literature, but there a variable is often just replicated several times to represent its value at different points in time. Since these points in time have to be fixed before the model is learned and deployed, a dilation or translation of events may easily misguide such a classifier.

The set of available variables for rule development is therefore altered in our approach and additionally depends on how far a rule has already been developed. At the beginning, one dichotomous variable per interval label is introduced. At this stage, the rule learner derives rules that consist of conditions on the existence or absence of certain intervals (labels) in the complete history, such as:

$$\textbf{if } \neg D \wedge A \wedge B \textbf{ then } \text{fault} \tag{1}$$

(reading 'if there is no interval labeled D but two intervals labeled A and B in the history, then we predict fault'). As soon as a rule requires the existence of at least two intervals in the history (here: A and B), additional variables are provided for further rule refinement. For each pair of such intervals a nominal attribute is introduced over the domain $2^{\mathcal{I}}$, indicating all relationships between A and B found in the history. In the next step the rule may now be specialized by adding yet another clause on the existence or absence of an interval but also by adding a condition on the relationship between two intervals, such as:

$$\textbf{if } \neg D \wedge A \wedge B \wedge A\langle R\rangle B \textbf{ then } \text{fault} \tag{2}$$

How to find this set R will be discussed in the next section. For instance, if $R = \{m, o\}$ the rule holds only for those cases where A *meets* or *overlaps* B. If a rule does not contain any condition on the relationship between two intervals k_i and k_j, this situation of complete ignorance corresponds to $R_{i,j} = \mathcal{I}$.

4.2 Sets of Relations

The newly introduced variables over the domain $2^{\mathcal{I}}$ have 2^{13} different values. For obvious reasons we do not want to check each of the 8192 possibilities to find the best rule specialization. To reduce this number we form groups of interval relationships and treat the relationships in a group collectively. If the intervals are derived in a preprocessing step from some other data, the exact location of the interval bounds is quite often questionable due to the several (heuristic) steps applied during preprocessing. In such a case it does not make much sense to consider relationships that require an exact match of interval bounds in full detail, because these matches are rather incidental. Here we are only interested in the relationships b, a, o, io, d, c but for reasons of completeness we form groups with the remaining relationships such that we obtain a partition such as:

$$\mathcal{G} = \{\{b, m\}, \{a, im\}, \{o\}, \{io\}, \{d\}, \{c\}, \{eq, s, is, f, if\}\} \tag{3}$$

While it is quite canonical to group m and b, it is less obvious what to do with *is* or *if*, for instance. Grouping $\{o, if\}$ and $\{io, f\}$ also appears to be a reasonable choice. For the remainder of the paper the actual partition is not important, it may be chosen differently from application to application.

Rather than considering all 2^{13} subsets of $\mathcal{I}$ we restrict ourselves to subsets of $\mathcal{G}$. More precisely, for $R_{i,j}$ we consider only subsets of $\mathcal{I}$ that can be represented as the union of elements of $\mathcal{G}$. For the remainder of this section by 'relationship r' we refer to *a group of relationships* $r \in \mathcal{G}$ rather than single element of $r \in \mathcal{I}$. Suppose that we have found a rule (1) requiring the existence of two intervals, say A and B. We want to extend the rule in the fashion of example (2), so our task is to find the set of relations R leading to the best refined rule according to some rule evaluation measure (e.g. the J-measure used in [8]). All necessary information for the rule evaluation measure can be found in the following contingency table, where the left/right part corresponds to the rule before/after the refinement on the interval relationship.

	base rule (1)		refinement (2)	
number of cases	class		class	
	positive	negative	positive	negative
rule antecedent holds	a	b	p_R	n_R
rule antecedent does not hold	c	d	$P - p_R$	$N - n_R$
total	P	N	P	N

Since the *base rule* is already given, the left half of the table is already known. The total number of positive and negative cases (P and N, resp.) remains identical for the base rule and the refinement, therefore it is sufficient to determine p_R and n_R for each possible refinement. To decide which subset R improves the given rule best, we need to construct the right contingency table for each possible choice of R. Thanks to the introduction of groups we have $2^7 = 128$ possibilities for R. This still requires some effort that we want to reduce further.

We restrict our discussion to the determination of n_R, but the same arguments hold analogously for p_R. While n_R denotes the number of sequences in which A and B can be observed in (at least) **one** of the relationships in R, we use m_R to denote the number of sequences in which A and B can be observed in **all** of the relationships in R simultaneously. Let us now examine how to determine n_R for the various choices of R:

- $|R| = 1$: We determine $n_{\{r\}}$ directly, that is, among the sequences that satisfy the antecedent of the base rule, we count in how many cases A and B satisfy relationship r.
- $|R| = 2$: Due to $|A \cup B| = |A| + |B| - |A \cap B|$ we have $n_{\{r,s\}} = n_{\{r\}} + n_{\{s\}} - m_{\{r,s\}}$. Since we know already the frequencies $n_{\{r\}}$, we additionally determine $m_{\{r,s\}}$ for all pairs of relationships r, s.
- $|R| = 3$: Repeating the same argument of the previous case, we have

$$n_{\{r,s,t\}} = n_{\{r\}} + n_{\{s\}} + n_{\{t\}} - m_{\{r,s\}} - m_{\{r,t\}} - m_{\{s,t\}} + m_{\{r,s,t\}}$$

  We only *approximate* n_R for this case, by neglecting the term $m_{\{r,s,t\}}$. We consider the error we introduce by this estimation to be rather small, because $m_{\{r,s,t\}}$ counts how often intervals A and B have been observed in *all three relationships r, s and t in the same sequence*.[1]

[1] If, however, this assumption is violated, we also determine $m_{r,s,t}$.

– $|R| > 3$: Instead of considering sets R with $|R| > 3$ we take $S = \mathcal{G} \backslash R$ into account, leading to $|S| \leq 3$. For two given intervals A and B, the condition $A\langle R \rangle B$ is equivalent to $\neg(A\langle S \rangle B)$. Therefore using the negation of a 'small' set S (of size ≤ 3) can be a substitute for using a 'large' set R (of size ≥ 4).[2] The frequency of $\neg(A\langle S \rangle B)$ can be readily obtained from $b - n_S$ (cf. contingency table), because b denotes the number of histories before rule refinement and n_S is the number of cases for which one relation $r \in S$ has been observed – consequently in $b - n_S$ cases no relationship $r \in S$ has been found in the sequences (but a relation $r \in R$).

By counting the frequencies n_R for $|R| = 1$ (7 counters when using (3)) and m_R for $|R| = 2$ (21 counters) we have reduced the number of frequency counts from 128 to 28. Only for sets R with cardinality 3 and 4 we thereby introduce a small error, in all other cases we obtain exact results.

Being able to derive the contingency table from these frequencies, we could in principle check for each possible value of R how the rule measure of the respective algorithm evaluates the refinement. Rather than trying all possibilities we use the following heuristic: The original rule is evaluated by some rule evaluation measure and yields a initial value of J_0 that any refinement must beat. We start from an empty set $R_0 = \emptyset$. For a given set R_i we choose the most promising relationship r such that the rule measure is optimized by $R_{i+1} = R_i \cup \{r\}$. This *optimization* is achieved for most rule measures by maximizing the difference $p_R - n_R$ from the contingency table. Among the sets of relationships $R_1 \subseteq R_2 \subseteq R_3$ (and $S_1 \subseteq S_2 \subseteq S_3$ for the negations) we finally choose the one that is evaluated best by the rule measure. In case this is J_0, no rule refinement is made. This represents, of course, a greedy optimization that does not guarantee global optimality.

5 Evaluation

Future work will include the application of this algorithm to medical data from health care. In this section, we apply our technique to a simple case in fault diagnosis. Figure 3 shows a small Java program using two parallel threads accessing a shared variable. The access to this variable is not synchronized, which is why some undesired effects occur. The first dummy thread calls methods f, g and h, one after the other. f stores a first value in the shared variable, g a second one, and h reads both values and returns their sum. We compare if the returned sum is correct (which gives us the classification label for a test run of the program). A second thread consists of a single call of k that also uses the shared variable temporarily, but at the end it restores the value from the beginning. A faulty result occurs, for instance, if the call of g overlaps the call of k. This program has been executed several times: each function call gives us an interval in our

[2] While this is perfectly true for two given intervals, when arguing about the relationship of A and B in a set of sequences there are little differences in the semantics, because multiple instances of A and B may occur.

interval sequence (the label is the name of the method, the execution time gives the temporal interval).

```
class A implements Runnable {
  Random random = new Random();
  private int[] shared;
  boolean ok;
  A(int[] data) { shared=data; }
  void f(int i) {
    Thread.sleep(random.nextInt(40));
    shared[0]=i;
  }
  void g(int i) {
    Thread.sleep(random.nextInt(40));
    shared[1]=i;
  }
  int h() {
    Thread.sleep(random.nextInt(40));
    int r = shared[0]+shared[1];
    return r;
  }
  public void run() {
    Thread.sleep(random.nextInt(40));
    int i,j;
    f(i=random.nextInt(10));
    g(j=random.nextInt(10));
    ok = (i+j==h()); // class label
  }
}
```

```
class B implements Runnable {
  Random random = new Random();
  private int[] shared;
  B(int[] data) { shared=data; }
  void k() {
    int i = shared[1];
    shared[1] = random.nextInt(1000);
    Thread.sleep(random.nextInt(40));
    shared[1] = i;
    return i;
  }
  public void run() {
    Thread.sleep(random.nextInt(100));
    k();
//  Thread.sleep(random.nextInt(40)); k();
  }
}
public class Demo {
  static int[] data = new int[2];
  public static void main(String[] args){
    ExecutorService app =
      Executors.newFixedThreadPool(2);
    app.execute(new A(data));
    app.execute(new B(data));
    app.shutdown();
  }
}
```

Fig. 3. Example Source Code from which interval sequences were generated: Calls to methods f, g, h or k are recorded, the history of a complete run is labeled with the value of the boolean variable 'ok' in class A.

To demonstrate the advantage of the proposed approach, the following table shows the *best* rule obtained when only single relationships ($|R| = 1$) are allowed (J-value of 0.1). A critical situation is given, if k covers the endpoint of either g or h. While this cannot be expressed exactly using a single relationship only, it is very well recovered from the top two rules delivered by the proposed algorithm – a great improvement in terms of the J-value is achieved. The third rule illustrates the usefulness of relationship negation: A fault will **not** occur if f occurs *before* k, which is expressed by the negative condition in the third rule.

single relation only	**if** $k \wedge g \wedge g\langle\{o\}\rangle k$ **then** fault	J=0.100
any number of relations	**if** $k \wedge g \wedge g\langle\{o\},\{d\}\rangle k$ **then** fault	J=0.205
	if $k \wedge h \wedge h\langle\{o\},\{d\}\rangle k$ **then** fault	J=0.139
	if $k \wedge f \wedge \neg(f\langle\{b,m\}\rangle k)$ **then** fault	J=0.078

A second dataset includes the interval k twice (uncomment last line of B.run()) and we generate rules that predict *no-fault* rather than *fault*. The best rule of our approach has a J-values more than twice as high as the best rule obtained when single relationships are allowed only. The first rule corresponds quite well to a negation of the top rule in the previous table. The second rule is an example for a rule that refines multiple relationships.

single relation	**if** $k \wedge g \wedge g\langle\{o\}\rangle k$ **then** fault	J=0.043
multiple relationships	**if** $g \wedge k \wedge \neg(k\langle\{io\},\{c\},\{eq,s,is,f,if\}\rangle g)$ **then** no-fault	J=0.091
	if $h \wedge f \wedge k\langle\{b,m\},\{o\}\rangle f \wedge k\langle\{b,m\}\rangle h$ **then** no-fault	J=0.074

6 Conclusion

Most of the work on mining sequential data deals with event sequences. We have argued in this paper, why interval sequences are better suited to capture the trace of variables over time. The few papers on mining interval sequences in the literature make use of Apriori-like algorithms. In this paper we have proposed a way to extend more traditional classifiers (such as rule or tree learners) to handle interval sequences. The resulting models are well suited to express temporal dependencies that are common in realistic applications, the expressiveness is greatly improved over earlier approaches.

References

[1] Agrawal, R., Srikant, R.: Mining sequential patterns. In: Proc. of 11th Int. Conf. on Data Engineering, Taipei, Taiwan (March 1995) 3–14

[2] Mannila, H., Toivonen, H., Verkamo, A.I.: Discovering frequent episodes in sequences. In: Proc. of the 1st ACM SIGKDD Int. Conf. on Knowl. Discovery and Data Mining, Menlo Park, Calif. (1995) 210–215

[3] Allen, J.F., Kautz, H.A., Pelavin, R.N., Tenenberg, J.D.: Reasoning about Plans. Morgan Kaufmann Publishers (1991)

[4] Höppner, F.: Discovery of temporal patterns – learning rules about the qualitative behaviour of time series. In: Proc. of the 5th Europ. Conf. on Principles of Data Mining and Knowl. Discovery. Number 2168 in LNAI, Freiburg, Germany, Springer (September 2001) 192–203

[5] Kam, P.S., Fu, A.W.C.: Discovering temporal patterns for interval-based events. In: Proc. of the 2nd Int. Conf. on Data Warehousing and Knowl. Discovery. Volume 1874 of LNCS., Springer (2000) 317–326

[6] Villafane, R., Hua, K.A., Tran, D., Maulik, B.: Knowledge discovery from series of interval events. Journal of Intelligent Information Systems **15**(1) (2000) 71–89

[7] Freksa, C.: Temporal reasoning based on semi-intervals. Artificial Intelligence **54**(1) (1992) 199–227

[8] Smyth, P., Goodman, R.M.: An information theoretic approach to rule induction from databases. IEEE Trans. on Knowledge and Data Engineering **4**(4) (August 1992) 301–316

[9] Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers (1993)