From: AAAI Technical Report WS-98-07. Compilation copyright © 1998, AAAI (www.aaai.org). All rights reserved.

Learning to Predict Rare Events in Categorical Time-Series Data

Gary M. Weiss[†] and Haym Hirsh

Department of Computer Science
Rutgers University
Piscataway, NJ 08855
gmweiss@att.com, hirsh@cs.rutgers.edu

Abstract

Learning to predict rare events from time-series data with non-numerical features is an important real-world problem. An example of such a problem is the task of predicting telecommunication equipment failures from network alarm data. For a variety of reasons, existing statistical and machine learning methods are not well suited to solving this class of problems. This paper describes *timeweaver*, a genetic algorithm based machine learning system that predicts rare events by identifying predictive temporal and sequential patterns within time-series data. Timeweaver is applied to two problems and is shown to produce results which are superior to existing learning methods.

Introduction

Time-series data is being generated and stored at an ever increasing pace. It is often useful to predict the future behavior of such a time-series. One example of such a prediction problem comes from the telecommunication industry. AT&T's long distance traffic is handled by 4ESS switches, which, when they detect a problem, send a timestamped alarm to a central site. Approximately 100,000 such alarms are generated each week and are routed to a rule-based expert system. This expert system relies on simple hand-crafted rules to identify the components that are likely to fail. Since these rules were created without any in-depth analysis of the data, we expect a machine learning approach to do a better job of predicting failures. We call this problem an *event prediction* problem since the task is to predict a future event (the failure of a piece of equipment) based on past events (the alarm messages).

In this paper we focus on *rare* event prediction problems with *categorical* features. The equipment failure prediction problem is such a problem because equipment failures occur very infrequently and because the alarm messages contain non-numerical features. Solving such problems is an extremely challenging task, since rare events are much harder to predict than common events and because existing statistical time-series prediction methods cannot handle data with categorical features. Predicting fraudulent credit card transactions and the start of transcription in DNA sequences are two additional problems with these characteristics.

Unfortunately, existing machine learning methods cannot be directly applied to these event prediction problems. Many machine learning methods are designed to solve concept learning problems, where the task is to induce a general description of a concept from specific instances of the concept. The conventional approach for solving event prediction problems has been to reformulate the problem into a concept learning problem. We argue that this approach is inappropriate since it loses important temporal and sequential information.

This paper describes *timeweaver* (Weiss 1998), a learning system designed to solve rare event prediction problems with categorical features by identifying predictive temporal and sequential patterns. Because it is generally not possible to make predictions with high accuracy for such problems, due do the inherent difficulty of the problem and the limited information in the observed data, timeweaver returns multiple solutions, which trade off precision and recall in different ways. Timeweaver also learns noise-tolerant rules, since learning to solve these difficult problems is similar to learning in a noisy environment. This contrasts with early work in the field which focused on finding sequence-generating rules from artificially generated, noise-free, datasets (Dietterich & Michalski 1985). Timeweaver can also be applied to sequence prediction problems, since a sequence can be viewed as a time-series where the timestamp is a sequence number.

Background

A time-series is a set of observations, or events, each recorded at a specific time and described by a fixed number of features. Classical time-series prediction involves predicting the next n successive observations from a history of past observations. These problems have been studied extensively within the field of statistics (Brockwell & Davis 1996), but statistical techniques are only applicable when the data is limited to numerical features. Neural

---

[†] Also AT&T Labs, Middletown NJ 07748

Copyright © 1998, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

networks using a "sliding window" technique or a recurrent network architecture have also been successfully applied to time-series prediction problems, but also require numerical features to be effective (Biggus 1996).

Previous efforts to solve event prediction problems with categorical features have relied on reformulating the problem into a concept learning problem. Dietterich & Michalski (1985) provide a general discussion of such methods. The reformulation process involves *transforming* the time-series data into an unordered set of examples by identifying which events go into each example (typically by sliding a window over the data), encoding these events as a single example and then determining the class value for each generated example. The transformation procedure will ensure that *some* of the temporal information from the time-series is preserved. Any concept learning program can then be applied to the transformed data. This approach has been used within the telecommunication industry to identify recurring transient network faults (Sasisekharan, Seshadri & Weiss 1996) and to predict catastrophic equipment failures (Weiss, Eddy, Weiss & Dube 1998).

In this paper we employ a *direct approach* for solving event prediction problems—one that does not require reformulating the problem. Previous work has also tried to solve similar problems using a direct approach. For example, work in computational learning theory has focused on learning sequences, regular expressions and pattern languages from data, but has resulted in few practical systems (Jiang & Li 1991; Lang 1992; Brazma 1993). Of greater relevance are data mining methods for identifying common temporal patterns in time-series data. Complete and relatively efficient algorithms exist for finding such patterns, but these common patterns are not guaranteed to be useful for prediction. Nonetheless, such algorithms have been used to identify regularities in telecommunication network alarm sequences in order to help predict future faults (Manilla, Toivonen & Verkamo 1995) and to find sequential patterns in a database of customer transactions (Agrawal & Srikant 1995).

The Event Prediction Problem

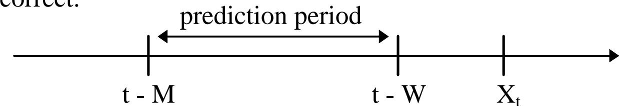
This section defines the event prediction problem, since our formulation differs in several key ways from the traditional time-series prediction problem.

Basic Problem Formulation

An event E_t is a timestamped observation which occurs at time t and is described by a set of feature-value pairs. An *event sequence* is a time-ordered sequence of events, $S = Et_1, Et_2, ..., Et_n$, which includes all n events in the time interval $t_1 \leq t \leq t_n$. Events are associated with a domain object D which is the source, or generator, of the events (if there are multiple domain objects then the definitions of event and event sequence are extended to specify the domain object). The *target event* is the event to be predicted and is specified by a particular set of feature-value pairs. The prediction problem is to learn a prediction procedure P that correctly predicts the target events. Thus, the prediction procedure is a function that maps an event sequence to a boolean prediction value which indicates whether to predict the target event. A prediction is made upon observation of each event, so P: $Et_1, Et_2, ..., Et_x \rightarrow \{+,-\}$, for each event Et_x.

This formulation can be applied to the telecommunication problem. Each alarm generated by the 4ESS switches is an event with three features: *device,* which identifies the component within the switch reporting the problem, *severity*, which identifies the severity of the problem and *code*, which specifies the exact problem. Each 4ESS switch is a domain object that generates an event sequence. The target event is any event with the problem code set to "FAILURE" and hence the problem is to learn a prediction procedure that correctly predicts the failure of a device.

To complete our formulation, we need to specify the semantics of a prediction. A *single positive prediction* is interpreted to mean that the target event will occur, regardless of any subsequent negative predictions. Thus, any action to be taken in response to such a prediction should occur immediately. A prediction period is associated with each target event, X_t, occurring at time t, as shown below. The warning time, W, is the "lead time" necessary for a prediction to be useful and the monitoring time, M, determines the maximum amount of time prior to the target event for which a prediction is considered correct.



A target event is *correctly* predicted if there is at least one positive prediction within its prediction period. A positive prediction is correct if it falls within the prediction period of some target event. Note that the warning and monitoring times should be set based on the problem domain. In general, however, the problem will be easier the smaller the value of the warning time and the larger the value of the monitoring time (however too large a value for the monitoring time will result in meaningless predictions).

Evaluation Measures

Our evaluation measures are summarized in Figure 1. The *recall* is the percentage of the target events correctly predicted. The *simple precision* is the percentage of the positive predictions that are correct. Simple precision is a misleading metric since it is inconsistent with how predictions are used—simple precision counts multiple positive predictions of the same target event multiple times. The *normalized precision* eliminates this multiple counting by replacing the number of correct positive predictions with the number of target events correctly predicted. However, this measure still does not account for the fact that n incorrect positive predictions located closely together may not be as harmful as the same number spread out over time (depending on the nature of the actions taken in response to

the prediction of a target event). *Reduced precision* remedies this. A prediction is "active" for a period equal to its monitoring time, since the target event should occur somewhere during that period. The reduced precision replaces the number of false positive predictions with the number of discounted false positives—the number of complete, non-overlapping, monitoring periods associated with a false prediction. Thus, two false positive predictions occurring a half monitoring period apart yields 1½ discounted false positives, due to an overlap in their active periods of ½ a monitoring period.

$$\text{Recall} \equiv \frac{\#\text{ Target Events Predicted}}{\text{Total Target Events}}, \quad \text{Simple Precision} \equiv \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$\text{Normalized Precision} \equiv \frac{\#\text{ Target Events Predicted}}{\#\text{ Target Events Predicted} + \text{FP}}$$

$$\text{Reduced Precision} \equiv \frac{\#\text{ Target Events Predicted}}{\#\text{ Target Events Predicted} + \text{Discounted FP}}$$

TP = True Positive Prediction FP = False Positive Prediction

Figure 1: Evaluation Measures for Event Prediction

The Basic Learning Method

Our learning method uses the following two steps:

1. *Identify prediction patterns*: Search the space of possible patterns to identify a set, C, of candidate prediction patterns. Each pattern $c \in C$ should do well at predicting a subset of the target events (i.e., recall may be traded off for higher precision).

2. *Generate prediction rules*: Generate an ordered list of prediction patterns from the set of candidate patterns from step 1. Various prediction rules can then be formed by creating a disjunction of the top *n* prediction patterns. These prediction rules can be used to form a prediction/recall curve, where the larger the value of *n*, the higher the recall (but presumably the lower the precision since the "best" rules are listed first).

This two step approach allows us to focus our effort on the more difficult task of identifying prediction patterns and makes it possible to generate multiple solutions with different precision/recall tradeoffs by combining prediction patterns in different ways. Also, by using a general search based method in the first step, we are able to easily incorporate our own evaluation metrics—which is necessary given the "non-standard" formulation of our prediction problem. For efficiency, our learning method also exploits the fact that for the class of problems we are interested in, target events occur infrequently. It does this by maintaining, for each prediction pattern, a boolean prediction vector of length *n* that indicates which of the *n* target events in the training set are correctly predicted. This information is used to ensure that a diverse set of patterns are identified and to help efficiently generate good prediction rules.

The Search Space

Our learning method requires a well defined space of prediction patterns. The language used for representing this space is similar to the language used for expressing the raw time-series data. A prediction pattern is a sequence of events in which consecutive events are connected by an ordering primitive, which defines sequential or temporal constraints between the events. The following ordering primitives define the ordering constraints:

- the *wildcard* "*" primitive matches any number of events so the prediction pattern A*D matches ABCD
- the *next* "." primitive matches no events so the prediction pattern A.B.C only matches ABC
- the *unordered* "|" primitive allows events to occur in any order and is commutative so that the prediction pattern A|B|C will match, amongst others, CBA.

The "|" primitive has highest precedence so the pattern "A.B*C|D|E" matches an A, followed immediately by a B, followed sometime later by a C, D and E, in any order. Each feature in the event is permitted to take on an additional feature value, "?", that matches any feature value. Each prediction pattern also includes a pattern duration. A prediction pattern *matches* a sequence of events within an event sequence if 1) the ordering constraints expressed in the prediction pattern are obeyed, 2) the events within the prediction pattern match events within the time-series, and 3) the events involved in this match occur within a period not exceeding the pattern duration. Once a match "completes", the target event is predicted. Note that for a prediction to be correct, it must only complete within the prediction period—the pattern may begin before the start of the prediction period. This language enables flexible and noise-tolerant prediction rules to be constructed, such as the rule: *if 3 (or more) A events and 4 (or more) B events occur within an hour, then predict the target event*. This language was designed to provide a small set of features useful for most real-world prediction tasks. In particular, this language does not include regular expressions and does not allow time intervals to be specified between individual events in the prediction patterns—the pattern duration only imposes a time constraint on the entire prediction pattern. Extensions to this language will require making only a few, very localized, changes to timeweaver.

A Genetic Algorithm-Based Method for Identifying Prediction Patterns

We use a genetic algorithm (GA) to identify a diverse set of prediction patterns. A GA was selected since most existing machine learning methods are not directly applicable and because the prediction problem translates naturally into a GA search problem. Furthermore, the *adaptive* nature of the GA should help with the search process—features in the pattern language which are not useful for prediction should

quickly die out, thereby reducing the effective size of the search space. The hope is that this approach, as opposed to the reformulation-based approach, will allow the characteristics of the problem space to influence the way the solutions are represented (based on the fact that the features in the pattern language define the representation).

The genetic algorithm is responsible for evolving a population of prediction patterns, where each individual should perform well at classifying a subset of the target events and which collectively should cover most of the target events. Thus, each individual represents only part of a complete solution. Our approach resembles that of classifier systems, which are GAs that evolve a set of classification rules (Goldberg 1989). The main difference between these approaches is that the rules in our approach are much simpler and cannot chain together (eliminating the need for credit assignment) and that instead of forming a ruleset from the entire population, a second step is used to form a ruleset from a subset of the rules in the population. Our approach is also similar to the approach taken by other genetic algorithms which learn disjunctive concepts from examples (Giordana, Saita & Zini 1994; McCallum & Spackman 1990). We use a steady-state GA instead of a generational GA because we expect the time to evaluate an individual to be large (due to large training sets) and a steady-state GA is believed to be more computationally efficient in this case. The main difference between these two types of GA's is that in a steady-state GA only a few individuals from the population change each "iteration". The basic steps in our steady-state GA are:

1. Initialize population
2. **while** stopping criteria not met
3. select 2 individuals from the population
4. apply crossover operator with probability P_C and mutation operator with probability P_M
5. evaluate the 2 newly formed individuals
6. replace 2 existing individuals with the new ones
7. **done**

Our population is initialized by creating prediction patterns containing a single event, where the feature values in this event are set 50% of the time to the wildcard feature value and the remaining time to a randomly selected feature value. The GA continues until either a pre-specified maximum number of iterations are executed or the performance of the population peaks. The first step within each iteration is to select two "fit" individuals. Next, either a crossover operator is applied to generate two new offspring from the selected individuals or a mutation operator is applied to each individual. Crossover is accomplished via a variable length crossover operator, as shown in Figure 3. Crossover points are randomly selected within each of the two individuals and the portion to the left of each crossover point is joined with the portion to the right of the other individual's crossover point. The lengths of the offspring may differ from that of the parents and hence over time prediction patterns of any size can be generated. The pattern duration of each child is set by trying several values (based on the parents' pattern durations) and then selecting the value which yields the best results. The ordering primitives are also "crossed over" in the crossover process.

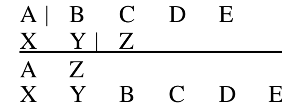


Figure 3: Variable Length Crossover

Our GA also employs mutation operators which randomly modify a prediction pattern. These mutation operators make the pattern more specific or general (by modifying the feature values or ordering primitives) and may also randomly change the prediction duration. The newly formed individuals are then evaluated on the entire training set and the evaluation metrics described earlier are computed. Two "unfit" individuals are then chosen from the population and are replaced by these new individuals.

The Selection and Replacement Strategy

The selection strategy is the most critical and complex component of the GA since it must balance two opposing criteria: it must focus the search for prediction patterns in the most profitable areas of the search space but must also maintain a diverse population. The challenge is to maintain a diverse population with only a minimal amount of global information that can be efficiently computed.

The fitness of an individual prediction pattern is based on both its precision and recall and is computed using the F-measure, defined below in equation 1, where β controls the importance of precision relative to recall (Van Rijsbergen 1979). Any fixed value of β yields a fixed bias in the algorithm and, in practice, leads to poor performance of the GA. To avoid this problem, for each iteration the value of β is randomly selected from the range of 0 to 1, similar to what was done by Murata & Ishibuchi (1995).

$$\text{fitness} = \frac{(\beta^2 + 1)\,\text{precision} \cdot \text{recall}}{\beta^2 \text{precision} + \text{recall}} \qquad \textbf{(1)}$$

The diversity maintenance strategy must ensure that a few prediction patterns do not dominate the population and that collectively the individuals in the population predict most, if not all, of the target events in the training set. We use a *niching* strategy called *sharing* to maintain a diverse population (Goldberg 1989). Diversity is encouraged by selecting individuals proportional to their *shared fitness,* where shared fitness is defined as fitness divided by niche count. The niche count, defined in equation 2, is a measure that indicates the degree of similarity of an individual *i* to the *n* individuals comprising the population.

$$\text{niche count}_i \equiv \sum_{j=1}^{n} (1 - \text{distance}(i,j))^3 \qquad \textbf{(2)}$$

The similarity of two individuals is measured using a phenotypic distance measure that measures the distance

based on the *performance* of the individuals. In our case this distance is simply the number of bit differences between the two individuals' prediction vectors (i.e., the number of target events for which they have different predictions). The more similar an individual to the rest of the individuals in the population, the smaller the distances and the greater the niche count value; if an individual is identical to every other individual in the population, then the niche count will be equal to the population size.

The replacement strategy also uses shared fitness. Individuals are chosen for deletion *inversely* proportional to their shared fitness, where the fitness component is computed by averaging together the F-measure of equation 1 with β values of 0, ½, and 1, so that patterns that perform poorly on precision *and* recall are most likely to be deleted.

Creating Prediction Rules

Finding the disjunction of prediction patterns which yields optimal performance on the training data is an NP-complete problem. To solve this problem efficiently, a greedy algorithm is used. This algorithm utilizes the information returned from the GA in the first step, which includes the precision and recall of each pattern as well as the prediction vector indicating which target events are correctly predicted. The algorithm for forming a solution, S, from a set of candidate patterns, C, is shown below:

```
1.  C = patterns returned from the GA; S = {};
2.  while C ≠ ∅ do
3.     for c ∈ C do
4.        if (increase_recall(S+c, S) ≤ THRESHOLD)
5.        then C = C - c;
6.        else c.score = PF × (c.precision - S.precision) +
7.                    increase_recall(S+c, S);
8.     done
9.     best = {c ∈ C, ∀x∈ C| c.score ≥ x.score}
10.    S = S || best; C = C - best;
11.    recompute S.precision on training set;
12. done
```

This algorithm incrementally builds solutions with increasing recall by heuristically selecting the "best" prediction pattern remaining in the set of candidate patterns, using the formula on lines 6 and 7 as an evaluation function. Prediction patterns that do not increase the recall of the solution by at least THRESHOLD are discarded. The evaluation function rewards those candidate patterns that have high precision and predict many of the target events not already predicted by S. The Prediction Factor (PF) controls the relative importance of precision vs. recall. Both THRESHOLD and PF affect the complexity of the learned concept and can be used to prevent overfitting of the data. This algorithm returns an ordered list of patterns, with the "best" (typically most precise) patterns at the beginning of the list. If *n* prediction patterns are returned, then *n* solutions are available: the first solution is comprised of the first prediction pattern in the list, the second solution the first two prediction patterns in the list, etc. Thus, a precision/recall curve can be constructed from S and the user can select a particular solution based on the relative importance of precision and recall.

Although the basic learning method only calls for the prediction rules to be formed upon termination of the GA, we also form prediction rules every 250 iterations of the GA. We do this for two, related, reasons. First, this allows us to accurately evaluate the true progress of the GA. This is necessary since the standard approach of basing the progress on the performance of the best individual in the population is not meaningful, given that each individual is only part of the total solution. Secondly, all of the prediction patterns used in these rules are saved to ensure than no good prediction patterns are ever lost. At the end of the execution of the GA, the final prediction rules are formed from these saved patterns. Saving these patterns is an optimization step and only leads to a moderate increase in timeweaver's performance.

This algorithm is quite efficient: if *n* is the number of candidate patterns returned from the first step (i.e., the population size), then the algorithm requires $O(n^2)$ computations of the evaluation function and $O(n)$ evaluations on the training data (step 11). Since all of the information required to compute the evaluation function is available and given the earlier assumption of large data sets and a relatively small number of target events, this leads to an $O(ns)$ algorithm, where *s* is the size of the training set. In practice, much less than *n* iterations of the for loop will be necessary, since the majority of the prediction patterns will not pass the test on line 4.

Experiments

This section describes the performance of timeweaver at predicting telecommunication equipment failures and at predicting the next command in a sequence of UNIX commands (the second prediction problem is to demonstrate that timeweaver can handle general event prediction problems). The default value of 1% for THRESHOLD and 10 for PF are used for all experiments. All results are based on evaluation on an independent test set, and, unless otherwise noted, on 2000 iterations of the GA. Precision is measured using reduced precision for the equipment failure problem, except in Figure 7 where simple precision is used to allow comparison with other approaches; for the UNIX command prediction problem, reduced and simple precision are identical, due to the nature of the prediction problem.

Predicting Equipment Failure

Most of the details of this prediction problem have already been provided and will not be repeated here. The problem is to predict telecommunication equipment failures from alarm data. The data contains 250,000 alarms reported from 75 4ESS switches, of which 1200 of the alarms indicate distinct equipment failures. Except when specified otherwise, all experiments have a 20 second warning time

and an 8 hour monitoring time. The alarm data was broken up into a training set with 75% of the alarms and a test set with 25% of the alarms (the training and test sets contain data from different 4ESS switches).

Figure 4 shows the performance of the learned prediction rules, generated at different points during the execution of the GA. The curve labeled "Best 2000" shows the performance of the prediction rules generated by combining the "best" prediction patterns from the first 2000 iterations. The figure shows that the performance improves with time. Improvements were not found after iteration 2000.

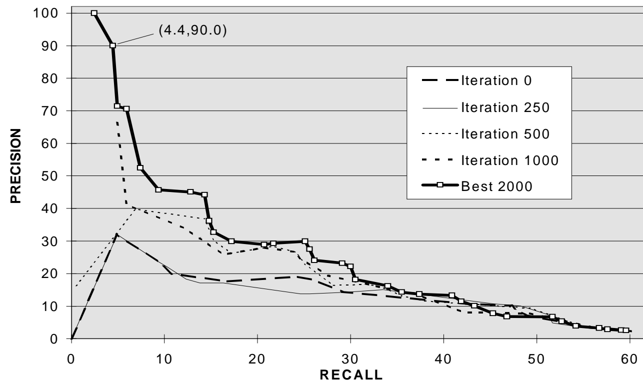


Figure 4: Learning to Predict Telecommunication Failures

These results are notable—the "baseline" strategy of predicting a failure every warning time (20 seconds) yields a precision of 3% and a recall of 63%. The curves generated by timeweaver all converge to this value, since the timeweaver essentially mimics this "baseline" strategy to maximize recall. A recall greater than 63% is never achieved since 37% of the failures have no events in their prediction period; this prevents correct predictions from being made. The prediction pattern corresponding to the first data point for the "Best 2000" curve in Figure 4 is: 351:<|TMSP|?|MJ|>*<|?|?|MJ|>*<|?|?|MN|>. This pattern indicates that a major severity alarm occurs on a TMSP device, followed sometime later by a major alarm and then by a minor alarm, all within a 351 second time period.

Experiments were run to vary the warning time and the results are shown in Figure 5. These results show that it is *much* easier to predict failures when only a short warning time is required. This effect is understandable since one would expect the alarms most indicative of a failure to occur shortly before the failure.

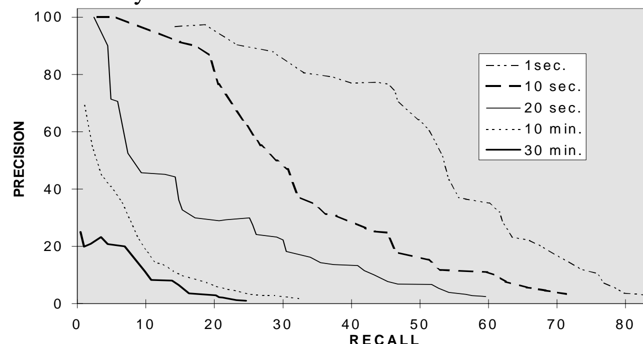


Figure 5: Effect of Warning Time on Learning

Figure 6 shows that increasing the monitoring time from 1 to 8 hours significantly improves timeweaver's ability to predict failures. The prediction problem should continue to become easier as the monitoring time increases past 8 hours, due to the increased prediction period. We believe this does not happen because the increased prediction period ensures there will be more patterns that will "predict" each target event, leading timeweaver to focus more of its attention on a large number of "spurious correlations" in the data.

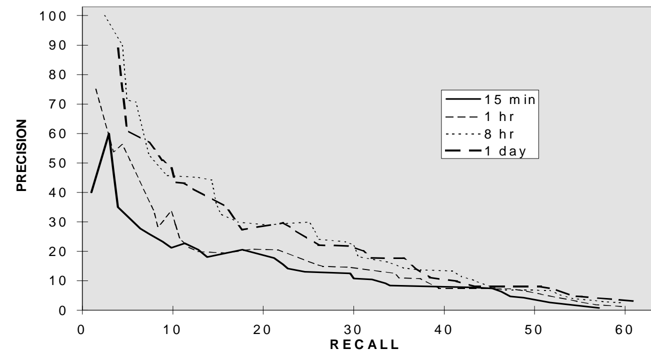


Figure 6: Effect of Monitoring Time on Learning

Comparison with Other Methods

The performance of timeweaver on the equipment failure prediction problem will now be compared against two rule induction systems, C4.5rules (Quinlan 1993) and RIPPER (Cohen 1995), and FOIL, a system that learns logical definitions from relations (Quinlan 1990). In order to use the "example-based" rule induction systems, the time-series data was first transformed by sliding a window of size *n* over the data and combining the *n* events within the window into a single example by "concatenating" the features. With a window size of 2, examples are generated with the features: device1, severity1, code1, device2, severity2 and code2. The classification assigned to each example is still based on the time-series data and the values of the warning and monitoring times. Since the equipment failures are so rare, the generated examples have an extremely skewed class distribution. As a result, neither C4.5rules nor RIPPER predict any failures when their default parameters are used. To compensate for the skewed distribution, various values of misclassification cost (i.e., the relative cost of false negatives to false positives) were tried and only the best results are shown in Figure 7. Note that in this figure, the number after the *w* indicates the window size and the number after the *m* the misclassification cost.

FOIL, which can learn from relations, is in many ways a more natural learning system for categorical time-series prediction problems, since it does not require any significant transformation of the data. With FOIL, the sequential information is encoded via the extensionally defined *successor* relation. Since FOIL provides no way for the user to modify the misclassification cost, the "default" value of 1 was used.

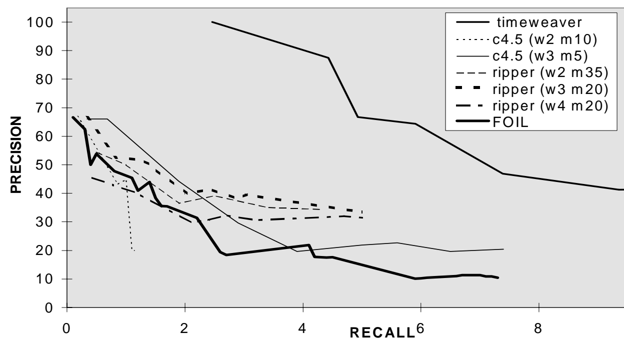


Figure 7: Comparison with Other ML Methods

C4.5rules required 10 hours to run for a window size of 3. RIPPER was significantly faster and could handle a window size up to 4; however, peak performance was achieved with a window size of 3. FOIL produced results which were generally inferior to the other methods. All three learning systems achieved only low levels of recall (note the limited range on the x-axis). For C4.5rules and RIPPER, increasing the misclassification cost beyond the values shown caused a single rule to be generated—a rule that always predicted the target event. Timeweaver produces significantly better results than these other learning methods and also achieves higher levels of recall.

Timeweaver can also be compared against ANSWER, the expert system responsible for handling the 4ESS alarms (Weiss, Ros & Singhal 1998). ANSWER uses a simple thresholding strategy to generate an *alert* whenever more than a specified number of interrupt alarms occur within a specified time period. These alerts can be interpreted as a prediction that the device generating the alarms is going to fail. Various thresholding strategies were tried and the thresholds generating the best results are shown in Figure 8. Each data point represents a thresholding strategy. Note that increasing the number of interrupts required to hit the threshold decreases the recall and tends to increase the precision. By comparing these results with those of timeweaver in Figure 4, one can see that timeweaver yields superior results, with a precision often 3-5 times higher for a given recall value (these results are off the scale in Figure 8). Much of this improvement is undoubtedly due to the fact that timeweaver's concept space is much more expressive than that of a simple thresholding strategy. This study also showed that ANSWER's existing threshold of 3 interrupts in 8 hours is sub-optimal.

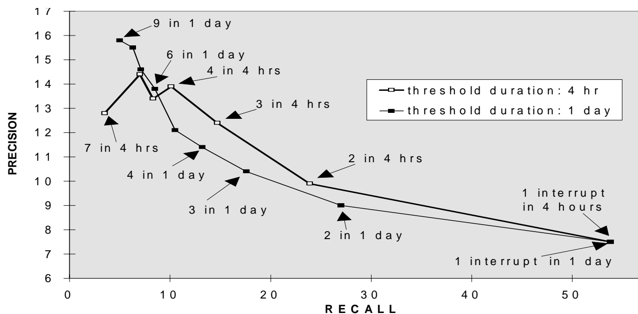


Figure 8: Using Interrupt Thresholding to Predict Failures

Predicting UNIX Commands

The task is to predict if the *next* UNIX command is a target command. The time at which each command was executed is not available, so this is a sequence prediction problem (thus the warning and monitoring times are both set to 1). The dataset contains 34,490 UNIX commands from a single user. Figure 9 shows the results for 4 target commands. Note that timeweaver does much better than the strategy of always guessing the target command (i.e., the strategy taken by timeweaver to achieve 100% recall). Timeweaver does better, and except for the *more* command much better, than a non-incremental version of IPAM, a probabilistic method that predicts the most likely next command based on the previous command (Davison & Hirsh 1998). The results from IPAM are shown as individual data points. The first prediction pattern in the prediction rules generated by timeweaver to predict the *ls* command is the pattern: 6:cd.?.cd.?.cd (the pattern duration of 6 means the match must occur within 6 events). This pattern will match the sequence cd ls cd ls cd, which is likely to be followed by another *ls* command.

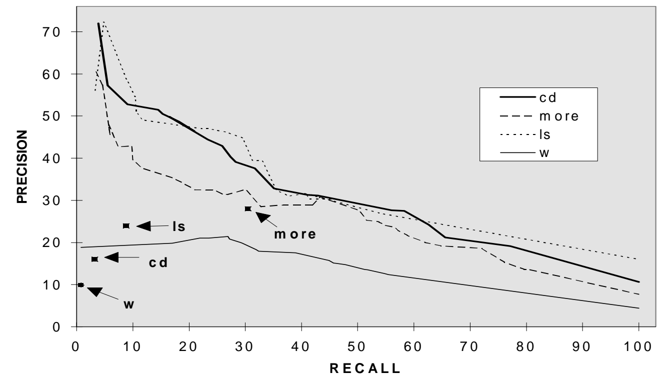


Figure 9: Predicting UNIX Commands

Future Work

The problem of predicting rare events within time-series data warrants additional reserach. We plan to further refine timeweaver and to apply it to additional problems. We also intend to further investigate the reformulation-based approach, in order to determine if more sophisticated transformations of the data significantly improve the efficacy of these methods and allow them to achieve performance comparable, or superior to, timeweaver. We believe that relational learning systems show particular promise. Although the performance of FOIL at predicting equipment failures was not particularly good, we feel its performance can be improved significantly by modifying FOIL to include a misclassification cost parameter and by supplying additional relations, beyond the simple *successor* relation. Finally, we will investigate the use of learning systems that permit relations to be defined intensionally. These systems will allow much more sophisticated relations to be provided than is possible with FOIL.

Conclusion

This paper investigated the problem of predicting rare events with categorical features from time-series data. Existing reformulation-based methods for solving such problems were shown to have several deficiencies. Most importantly, information is lost in the reformulation process and the resulting problem formulation does not really fit the original problem (e.g., predictive accuracy is not a good evaluation metric for the prediction problem). This paper showed how the rare event prediction problem could be properly formulated as a machine learning problem and how timeweaver, a GA-based machine learning program, could solve this class of problems by identifying predictive temporal and sequential patterns *directly* from the unmodified time-series data.

References

Agrawal, R., and Srikant, R. 1995. Mining sequential patterns. In Proceedings of the International Conference on Data Engineering.

Biggus, J. P. 1996. *Data Mining with Neural Networks*. McGraw Hill.

Brazma, A. 1993. Efficient Identification of Regular Expressions from Representative Examples. In Proceedings of the Sixth Annual Workshop on Computational Learning Theory, 236-242.

Brockwell, P. J., and Davis, R. 1996. *Introduction to Time-Series and Forecasting.* Springer-Verlag.

Cohen, W. 1995. Fast Effective Rule Induction. In Proceedings of the Twelfth International Conference on Machine Learning, 115-123**.**

Davison, B., and Hirsh, H. 1998. Probabilistic Online Action Prediction. In Proceedings of the AAAI Spring Symposium on Intelligent Environments.

Dietterich, T., and Michalski, R. 1985. Discovering patterns in sequences of Events, *Artificial Intelligence*, 25:187-232.

Giordana, A., Saitta, L., and Zini, F. 1994. Learning Disjunctive Concepts by Means of Genetic Algorithms. In Proceedings of the Eleventh International Conference on Machine Learning, 96-104.

Goldberg, D. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley.

Jiang, T., and Li, M. 1991. On the complexity of learning strings and sequences. In Proceedings of the Fourth Annual Workshop on Computational Learning Theory, 367-371. Santa Cruz, CA: Morgan Kaufmann.

Lang, K. 1992. Random DFA's can be approximately learned from sparse uniform examples, In Proceedings of the Fifth Annual Workshop on Computational Learning Theory.

Manilla, H., Toivonen, H., and Verkamo, A. 1995. Discovering Frequent Episodes in Sequences. In First International Conference on Knowledge Discovery and Data Mining, 210-215, Montreal, Canada, AAAI Press.

McCallum, R., and Spackman, K. 1990. Using genetic algorithms to learn disjunctive rules from examples. In Proceedings of the Seventh International Conference on Machine Learning, 149-152.

Murata, T., and Ishibuchi, H. 1995. MOGA: Multi-Objective Genetic Algorithms. In IEEE International Conference on Evolutionary Computation, 289-294.

Quinlan, J. R., 1990. Learning Logical Definitions from Relations, *Machine Learning*, 5: 239-266.

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann.

Sasisekharan, R., Seshadri, V., and Weiss, S. 1996. Data mining and forecasting in large-scale telecommunication networks, *IEEE Expert*, 11(1): 37-43.

Van Rijsbergen, C. J. 1979. *Information Retrieval*, Butterworth, London, second edition.

Weiss, G. M. 1998. Timeweaver WWW Home Page: http://paul.rutgers.edu/~gweiss/thesis/timeweaver.html.

Weiss, G. M., Eddy, J., Weiss, S., and Dube., R. 1998. Intelligent Technologies for Telecommunications. In *Intelligent Engineering Applications*, Chapter 8, CRC Press.

Weiss, G. M., Ros J. P., and Singhal, A. (1998). ANSWER: Network Monitoring using Object-Oriented Rules. In Proceedings of the Tenth Conference on Innovative Applications of Artificial Intelligence, Madison, Wisconsin.