Timeweaver: a Genetic Algorithm for Identifying Predictive Patterns in Sequences of Events

Gary M. Weiss

Rutgers University and AT&T Labs
101 JFK Parkway
Short Hills, NJ 07078

Abstract

Learning to predict future events from sequences of past events is an important, real-world, problem that arises in many contexts. This paper describes Timeweaver, a genetic-based machine learning system that solves the event prediction problem by identifying predictive temporal and sequential patterns within data. Timeweaver is applied to the task of learning to predict telecommunication equipment failures from 250,000 alarm messages and is shown to outperform existing methods.

1 INTRODUCTION

Data is being generated and stored at an ever-increasing pace, and, partly as a consequence of this, there has been increased interest in how machine learning and statistical techniques can be employed to extract useful knowledge from this data. When this data is time-series data, it is often important to be able to predict future behavior based on past data. In this paper we are interested in the problem of predicting specific types of rare future events, which we refer to as *target* events, from sequences of timestamped events. We restrict ourselves to domains where the events are described by categorical (i.e., non-numerical) features, since statistical methods already exist that can solve time-series prediction problems with numerical features. We call the class of problems we address in this paper *rare event prediction problems*. For these prediction problems, every time an event is received, a prediction procedure is applied which, based on the past events, determines whether the target event will occur in the near future. The problem of predicting telecommunication equipment failures from logs of alarm messages is one example of this type of prediction problem. In this case, prediction of a failure might cause one to replace, or at least route phone traffic around, the suspect piece of equipment. Other examples of rare event prediction problems include predicting fraudulent credit card transactions and the start of transcription in DNA sequences.

Machine learning and statistical methods have been used to solve problems similar to the rare event prediction problem, but most of these methods are not applicable to this class of problems. The statistical methods do not apply because they require numerical features (Brockwell & Davis, 1996). The many machine learning methods that perform classification do not apply because they assume unordered examples—not time-ordered events. Thus, these methods cannot learn from sequential or temporal relationships *between* events. The machine learning methods that are useful for modeling sequences are also not appropriate, since we do not need to model the entire sequence—we only need to predict one specific type of event within a window of time.

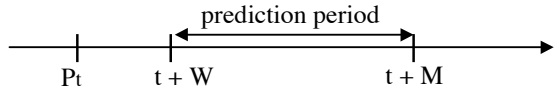
Our approach to solving the event prediction problem involves using a genetic algorithm to *directly* search for predictive patterns in the data. Our system will learn a set of rules of the form pattern $\Rightarrow$ target event, and hence may be considered a classifier system. However, because our system does not provide any form of internal memory or chaining of rules, and because the rules all have a common right-hand side, it is more appropriate to view our system as a genetic-based machine learning system. We feel our work shares much in common with other such systems, most notably COGIN (Greene & Smith, 1993) and GABIL (De Jong, Spears & Gordon, 1993).

The event prediction problem has been described in an earlier paper, but only a very brief description of the genetic algorithm was provided (Weiss & Hirsh, 1998). In this paper we provide a detailed description of Timeweaver, our genetic-based machine learning system that solves the rare event prediction problems by identifying predictive patterns in the data. The event prediction problem has some interesting characteristics that affect the design of our genetic algorithm. First, because we expect to encounter very large datasets, the GA must minimize the number of patterns that it evaluates. Second, because the events we are interested in predicting are typically rare and difficult to predict, predictive accuracy is not an appropriate fitness measure—the strategy of never predicting any target events would often maximize predictive accuracy. To avoid this problem, we base our fitness function on recall and precision, two measures from the information

retrieval literature. Recall will measure the percentage of target events that are successfully predicted and precision the percentage of predictions that are correct. By factoring in recall, we can ensure that a prediction strategy is developed that predicts the majority of the target events.

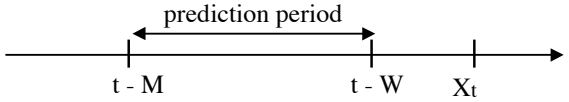
2 THE EVENT PREDICTION PROBLEM

In this section we provide a formal description of the event prediction problem. An event E_t is a timestamped observation occurring at time *t* that is described by a set of feature-value pairs. An *event sequence* is a time-ordered sequence of events, $S = E_{t1}, E_{t2}, ..., E_{tn}$. The *target event* is the specific type of event we would like to learn to predict. The event prediction problem is to learn a prediction procedure that, given a sequence of timestamped events, correctly predicts whether the target event will occur in the "near future". In this paper we assume the prediction procedure involves matching a set of learned patterns against the data and predicting the occurrence of a target event if the match succeeds. We say a prediction occurring at time t, P_t, is *correct* if a target event occurs within its *prediction period*. As shown below, the prediction period is defined by a warning time, W, and a monitoring time, M.



The warning time is the "lead time" necessary for a prediction to be useful and the monitoring time determines how far into the future the prediction extends. While the value of the warning time is critically dependent on the problem domain (e.g., how long it takes to replace a piece of equipment), there is generally a lot of flexibility in assigning the monitoring time. The larger the value of the monitoring time, the easier the prediction problem, but the less meaningful the prediction.

A target event is correctly predicted if *at least one* prediction is made within its prediction period. The prediction period of target event X, occurring at time t, is shown below:



3 THE GENERAL APPROACH

Our approach to solving the event prediction problem involves two steps. In the first step, a genetic algorithm is used to search the space of prediction patterns, in order to identify a set of patterns that individually do well at predicting a subset of the target events and collectively predict most of the target events. This is a Michigan-style GA, since each individual pattern is only part of a complete solution. Other GA-based systems have used this approach to learn disjunctive concepts from examples (Giordana, Saitta & Zini, 1994; Greene & Smith, 1993; McCallum & Spackman, 1990). However, rather than simply forming a solution from all of the individuals in the population, we employ a second step. This step orders the patterns from best to worst, based primarily on the precision of their predictions, and prunes redundant patterns (i.e., those patterns that do not predict any target events not already predicted by some "better" pattern). A family of prediction strategies is then formed by incrementally adding one pattern at a time and a precision/recall curve is generated based on these strategies. A user can then use this curve, and the relative cost of false predictions versus missed (i.e., not predicted) target events, to determine the optimal prediction strategy.

Our GA uses steady-state reproduction, where only a few individuals are replaced at once, rather than generational reproduction, where a significant percentage of the population is replaced. We chose to use steady-state reproduction because it makes newly created members immediately available for use, whereas with generational reproduction, the new individuals cannot be used until the next generation (Syswerda, 1990). Given that large datasets will make it computationally expensive to evaluate each new pattern, we believe it is very important that new individuals are made immediately available.

4 THE GENETIC ALGORITHM

This section describes a genetic algorithm that searches for patterns that successfully predict the future occurrence of target events. The basic steps in our steady-state GA are shown below:

1. Initialize population
2. **while** stopping criteria not met
3. select 2 individuals from the population
4. apply the 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**

Since we are using a Michigan-style GA, the performance of the population cannot be accurately estimated based on the performance of individual members of the population. Evaluating the performance based on all of the patterns would also be misleading, since there may be some very poor rules in the population. To accurately estimate the performance of the GA, we apply our "second step" every 250 iterations of the GA. This forms a complete prediction strategy using the best patterns in the population (as described in detail in Section 5).

The remainder of this section describes the most important aspects of the GA. In particular, much attention is devoted to the fitness function and diversity maintenance strategy, since these account for much of the complexity of our system. We use a niching strategy called sharing to ensure that diversity is preserved and that the prediction patterns in the population cover a majority of the target events. The niching strategy results in a new measure, *shared fitness*, which factors in both fitness and diversity considerations. It is this shared

fitness measure that is used to implement the selection and replacement strategies. In particular, individuals are selected proportional to their shared fitness and removed inversely proportional to their shared fitness.

4.1 REPRESENTATION

Each individual in our GA is a prediction pattern—a pattern used to predict target events. The language used to describe these patterns is similar to the language used to represent the raw event sequences. A prediction pattern is a sequence of events in which consecutive events are connected by an ordering primitive that defines ordering constraints between these events. The following ordering primitives are supported:

- 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 *parallel* "|" 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 an event is permitted to take on any of a predefined list of valid feature values, as well as the wildcard ("?") value, which matches any feature value. For example, if events in a domain are described by three features, then the event <?, ?, b> would match any event in which the third feature has the value "b". Finally, each prediction pattern also includes a *pattern duration*. A prediction pattern *matches* a sequence of events within an event sequence if: 1) the events within the prediction pattern match events within the event sequence, 2) the ordering constraints expressed in the prediction pattern are obeyed, and 3) the events in the event sequence involved in this match occur within a period not exceeding the pattern duration. Once a match succeeds, a target event is predicted.

This language enables flexible and noise-tolerant prediction rules to be constructed, such as the rule: *if 2 (or more) A events and 3 (or more) B events occur within an hour, then predict the target event*. This rule can be expressed using the pattern "1 hour: A|A|B|B|B", or any permutation of this pattern, such that there are a total of 2 A's and 3 B's. This language was designed to provide a small set of features useful for many real-world prediction tasks. In particular, this language does not include regular expressions nor does it allow time intervals to be specified between individual events in the prediction patterns. However, extensions to this language would require only a few, very localized, changes to Timeweaver.

Prediction patterns are encoded as variable length integer strings in our GA. If f features are used to describe each event, then each event is encoded using f integers; each feature value is mapped to an integer value when the data is read into our system. A prediction pattern with n events, each described by f features, is represented by a string containing $n(f+1)+1$ integers, since there is one integer-valued ordering primitive per event and one pattern duration per prediction pattern.

4.2 INITIALIZATION OF THE POPULATION

The population is initialized by generating prediction patterns which contain only a single event, where the feature values in this event are set 50% of the time to the wildcard value and the remaining time to a randomly selected valid feature value. The patterns are generated in this manner so that they will not start off overly specific—in which case they might not match any events in the training data, which would prevent the GA from effectively exploring the search space. The crossover operator, described in the next section, will allow these single-event patterns to grow, as necessary. A future enhancement might be to seed some of the initial patterns based on the training data—although this approach could overly bias the search.

4.3 GENETIC OPERATORS

Timeweaver employs a crossover and mutation operator. Crossover is accomplished via a variable length crossover operator, as shown in Figure 1. The first parent, P1, has a prediction pattern with 4 events and the second parent, P2, has one with 2 events. Each event contains two features and the ordering primitive is stored in the third position. For each parent, an event within each pattern is chosen at random and then a single, shared, intra-event offset is selected, to specify the point *within* the events at which the crossover takes place (in Figure 1 this occurs after the first feature). Then, the portion to the left of each crossover point is joined with the portion to the right of the other individual's crossover point. This variable-length crossover operator allows the lengths of the offspring to differ from that of their parents, so that over time the GA can generate prediction patterns of any size.

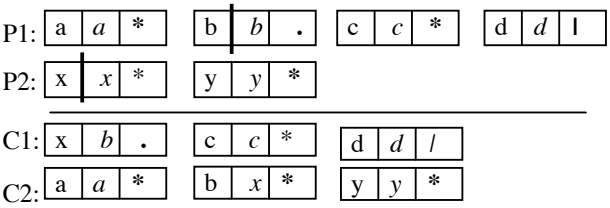


Figure 1: Variable Length Crossover

The pattern duration associated with each pattern is also involved in the crossover process, but is handled in a different manner. Each child is evaluated on the training data using the pattern duration from P1, P2, and the average of these two values. The pattern duration yielding the best fitness, d, is then tentatively selected. If d is the average of P1 and P2 then the child's pattern duration is set to this value; otherwise, one additional value is evaluated before the best value is selected for the

child. The additional value is chosen as follows: if d is the smaller pattern duration of P1 and P2, then a value randomly chosen between 0 and d is evaluated; otherwise a random value between d and $2d$ is evaluated. This procedure allows new pattern duration values to be introduced.

The mutation operator modifies the value of either the pattern duration, the feature values in the events, or the ordering primitives. In each case, mutation causes a valid random value to be selected. Note that this allows more general or specific patterns to be formed. For example, when a feature value is mutated, 50% of the time it will be set to the wildcard value and 50% of the time to a randomly selected feature value. Mutation of a pattern duration with value d results in a random value between 0 and $2d$ being generated.

4.4 THE FITNESS FUNCTION

The fitness function must take an individual pattern and return a value indicating how good the pattern is at predicting target events. As mentioned earlier, predictive accuracy is not a good measure of fitness since the target events may occur very infrequently and because we are using the Michigan approach where each pattern is only part of the total solution. Consequently, recall and precision form the basis of our fitness function. These information retrieval measures, which are described in detail below, are appropriate for measuring the fitness of individual patterns or collections of patterns. These measures are summarized in Figure 2. For our problem, *recall* is the fraction of the target events that are successfully predicted and *precision* is the fraction of the (positive) predictions that are correct. The negative predictions are not factored in because they are always ignored—only the presence or absence of a positive prediction is used to determine whether to predict the future occurrence (or non-occurrence) of a target event.

As described in Section 2, a target event may be predicted whenever an event is received; thus, multiple valid predictions of a single target event may occur. Precision is a misleading measure since it counts these multiple predictions multiple times. Since the user is expected to take action upon seeing the *first* positive prediction, counting the subsequent predictions in the precision measure is improper. The *normalized precision* eliminates this multiple counting by replacing the number of correct (positive) predictions with the number of target events correctly predicted.

Normalized precision 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). We use *reduced precision* to remedy this. A prediction is considered "active" for a period equal to its monitoring time, since the target event should occur somewhere during that period. Two false predictions located close together will have overlapping active periods. The reduced precision measure replaces the number of false positive predictions with the number of complete, non-overlapping, prediction periods associated with a false prediction. Thus, two false predictions occurring a half-monitoring period apart yields 1½ "discounted" false positives. For the remainder of this paper, the term precision will refer to reduced precision.

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

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

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

TP = True Positive Prediction FP = False Positive Prediction

Figure 2: Evaluation Measures for Event Prediction

The fitness of a pattern is based on both its precision and recall. However, there are many ways in which these two measures can be combined to form the fitness function. Extensive testing was done on synthetic data to evaluate alternative strategies for combining these two measures. Different fixed weighting schemes were tried, but all yielded poor results. Typically, the resulting population of patterns contained either very precise patterns that each covered only a few target events or very imprecise patterns that covered many of the target events. That is, the GA tended to optimize for precision or recall, but not both, no matter how the relative weighting of these two measures was adjusted. Even the strategy of progressively increasing the relative importance of precision did not eliminate this problem. These fitness functions even performed poorly on synthetic data where a single pattern existed that yielded 100% precision and 100% recall. Thus, it is clear that these fitness functions prevent us from effectively searching the search space.

The solution we adopted is to modify the relative importance of precision and recall after each iteration of the GA. Specifically, we use an information retrieval measure known as the F-measure, which is defined below in equation 1 (Van Rijsbergen, 1979). The value of β, which controls the relative importance of precision to recall, was changed each iteration of the GA, so that it cycles through the values between 0 and 1 using a step-size of .10.

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

4.5 DIVERSITY MAINTENANCE STRATEGY

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. Our challenge is to maintain diversity without making the search too unfocused and to assess diversity efficiently, using a minimal amount of global information that can be

efficiently computed. 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,* a measure that factors in an individual's fitness as well as its similarity to other individuals in the population. The degree of similarity of an individual i to the n individuals comprising the population is measured by the niche count, defined in equation 2. Experiments using synthetic data led us to choose a value of 3 for α.

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

The similarity of two individuals is measured using a phenotypic distance measure that measures distance based on the performance of the individuals at predicting target events. The performance of each individual is represented by a Boolean prediction vector of length t that indicates which of the t target events in the training set the individual correctly predicts. The distance between two individuals is the fraction of the target events for which the two individuals have different predictions, which can be computed from the number of bit differences in the prediction vectors. 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 *shared fitness* is defined as the fitness divided by the niche count.

Note that the distance measure focuses on which target events are successfully predicted—false predictions are not represented in the prediction vectors. This is not a major concern because false predictions are already penalized in the fitness function. Our approach exploits the fact that for many applications the target events are rare, which greatly reduces the time required to compute the distance measures.

We can calculate the number of bit-wise prediction vector comparisons required in order to keep the niche counts current. Since two new individuals are introduced into the population each iteration, we must calculate the niche counts of these two new individuals from scratch each iteration, and *update* the niche counts of the remaining individuals, due to the changes in the population. Assuming there are P individuals in the population, the niche count for each of the two new individuals requires P-1 prediction vector comparisons, since each pattern must be compared to all other patterns. The niche counts of the remaining individuals can be incrementally updated by considering, for each individual, the number of new bit differences added/eliminated as a result of replacing two old individuals with two new individuals. This incremental update requires 4 prediction vector comparisons per individual. Thus, a total of 2(P-1) + 4(P-2), or 6P-10, prediction vector comparisons are required each iteration, instead of the P(P-1) that would be required without the incremental updates. For the domains we have investigated, the time required to do these updates is just a small fraction of the time required to evaluate the new individuals.

As mentioned earlier, new individuals are selected with a probability proportional to their shared fitness, where the relative value of precision and recall depends on the value of β for that iteration. The replacement strategy also uses shared fitness, but in this case individuals are chosen *inversely* proportional to their shared fitness. Furthermore, for replacement, 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 both precision *and* recall are most likely to be deleted.

5 CREATING PREDICTION RULES

This section describes an efficient algorithm for ordering the prediction patterns returned by the genetic algorithm and pruning "redundant" patterns. The algorithm utilizes the precision, recall, and prediction vector returned by the GA for each pattern. 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∈Cl c.score ≥ x.score}
10.     S = S ll 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, in order to prevent overfitting of the data. This step will also remove "redundant" patterns that do not predict any target events not already predicted. 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 and recall, and increasing it will reduce the complexity of the learned prediction rule. Experimental results indicate that selecting patterns solely based on their precision yields results only slightly worse than those produced using the current evaluation function, where PF is set to 10.

This algorithm is quite efficient: if p candidate patterns are returned by the GA (i.e., p is the population size), then the algorithm requires $O(p^2)$ computations of the evaluation function and $O(p)$ evaluations on the training data (step 11). If we assume that there is much more training data than individuals in the population and that

target events are rare (this affects the time to compute the evaluation function), then the running time of the algorithm is bounded by the time to evaluate the patterns on the training data, which is O(*ps*), 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.

6 RESULTS

This section describes the results of applying Timeweaver to the problem of predicting telecommunication equipment failures and predicting the next command in a sequence of UNIX commands. 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 5 where "simple" precision is used to allow comparison with other approaches. For the UNIX command prediction problem, reduced and simple precision are identical, since the monitoring period is equal to 1.

6.1 PREDICTING EQUIPMENT FAILURES

The problem is to predict telecommunication equipment failures from historical 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 (each data set contains alarms from different 4ESS switches). Figure 3 shows the performance of the learned prediction rules, generated at different points during the execution of the GA. The curve labeled "Best 2000" was 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.

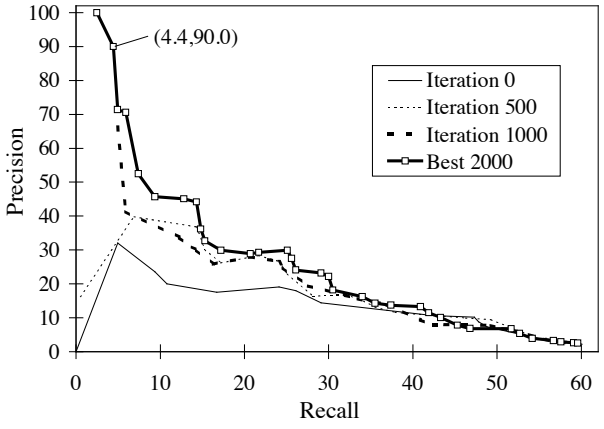


Figure 3: Learning to Predict Equipment Failures

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 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. The prediction pattern corresponding to the first data point for the "Best 2000" curve in Figure 3 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 interval.

Experiments were run to vary the warning and monitoring times, in order to assess the sensitivity of the problem to these problem parameters. The results for varying the warning time, shown in Figure 4, demonstrate 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. The problem was not nearly as sensitive to the value of the monitoring time. Increasing this value led only to modest improvements in precision until the value reached hours—beyond which only minimal improvements were seen.

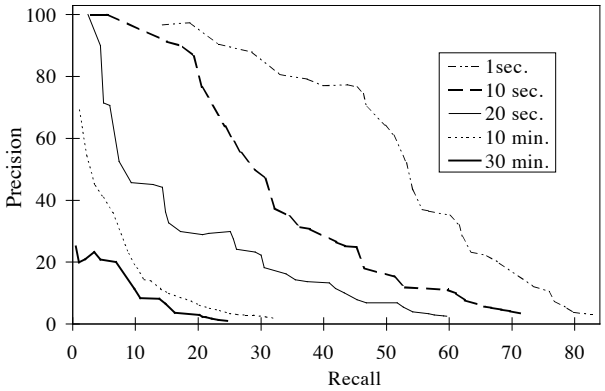


Figure 4: Effect of Warning Time on Learning

6.2 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 Horn clauses from ground literals (Quinlan, 1990). In order to use the "example-based" rule induction systems, the event sequence data is 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 event sequence 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 predicts 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 5. Note that in Figure 5 the number after the *w* indicates the window size and the number after the *m* the misclassification cost.

FOIL is a more natural learning system for event prediction problems, since it can represent sequence information using relations such as *successor*(E1, E2) and *after*(E1, E2), and therefore does not require any significant transformation of the data. FOIL provides no way for the user to modify the misclassification cost, so the "default" value of 1 is used.

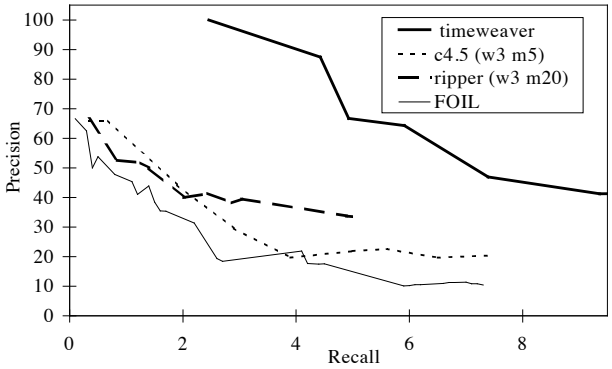


Figure 5: Comparison with Other ML Methods

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 that 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 6. 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 3, one can see that Timeweaver yields superior results, with a precision often 3-5 times higher for a given recall value. Much of this improvement is undoubtedly due to the fact that Timeweaver's pattern language is much more expressive than a simple thresholding language.

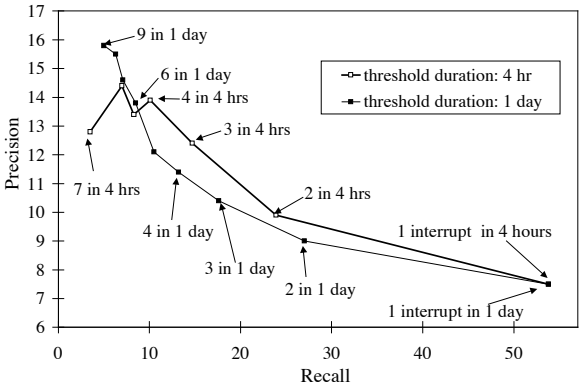


Figure 6: Using Thresholds to Predict Failures

6.3 PREDICTING UNIX COMMANDS

In order to demonstrate the effectiveness of our system across multiple domains, we applied our system to the task of predicting whether the *next* UNIX command in a log of commands is the 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). Note that Timeweaver can solve sequence prediction tasks because they may be considered a special case of an event prediction task. The dataset contains 34,490 UNIX commands from a single user. Figure 7 shows the results for 4 target commands. 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 strategy 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 matches the sequence cd ls cd ls cd, which is likely to be followed by another *ls* command.

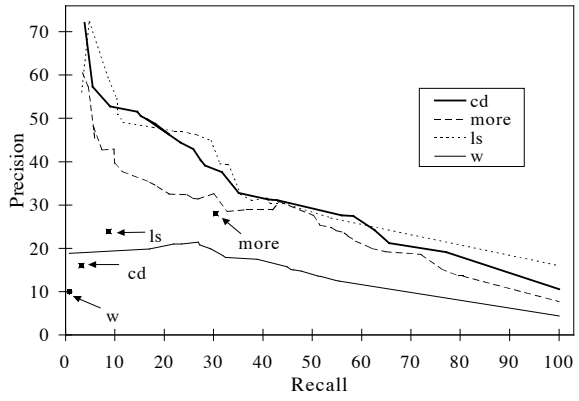


Figure 7: Predicting UNIX Commands

7 RELATED RESEARCH

Our system performs prediction, which may be viewed as a type of classification task, and hence our system shares much in common with classifier systems (Goldberg, 1989). However, due to the simplicity of our rules, we feel our work has more in common with other genetic-based systems that use similarly simple rules to classify examples (Greene & Smith, 1993; De Jong, Spears & Gordon, 1993; McCallum & Spackman, 1990), as well as evolutionary methods that build decision trees to classify examples (Marmelstein & Lamont, 1998; Ryan & Rayward-Smith, 1998).

Transforming the event sequence data into an unordered set of examples permits existing concept-learning programs to be used (Dietterich & Michalski, 1985). These programs handle categorical features and we used this technique in Section 6 so that C4.5 and RIPPER, two popular machine learning program, could be applied to the event prediction problem. This approach has also 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, 1998). The problem with these techniques is that the transformation process will lose some sequence and temporal information and one does not know aprioi what information is useful.

8 CONCLUSION

This paper investigated the problem of predicting rare events with categorical features from event sequence data. We showed how the rare event prediction problem could be formulated as a machine learning problem and how Timeweaver, a genetic-based machine learning system, could solve this class of problems by identifying predictive temporal and sequential patterns *directly* in the unmodified event sequence data. This approach was compared to other machine learning approaches and shown to outperform them.

Acknowledgments

Thanks to members of the Rutgers machine learning research group, and especially Haym Hirsh, for feedback on this work.

References

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*.

De Jong, G. A., Spears, W. M., and Gordon, D. 1993. Using Genetic Algorithms for Concept Learning. *Machine Learning*. 13:161-188.

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.

Greene, D. P., and Smith, S. F. 1993. Competition-Based Induction of Decision Models from Examples. *Machine Learning*. 13: 229-257.

Marmelstein, R, and Lamont, G. 1998. Pattern Classification using a Hybrid Genetic Program-Decision Tree Approach. In *Proceedings of the Third Annual Genetic Programming Conference*, 223-231.

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.

Neri, F. & Saitta, L., 1996. An analysis of the Universal Selection Suffrage Operator. *Evolutionary Computation*, 4(1): 87-107.

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

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

Ryan, M. D., and Rayward-Smith, V. J. 1998. The evolution of decision trees. In *Proceedings of the Third Annual Genetic Programming Conference*, 350-358.

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

Syswerda, G. 1990. In the *First Workshop on the Foundations of Genetic Algorithms and Classification Systems*, Morgan Kaufmann.

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

Weiss, G. M., Eddy, J., Weiss, S., and Dube., R. 1998. Intelligent Technologies for Telecommunications. In *Intelligent Engineering Applications*, Chapter 8, CRC Press, 249-275.

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.

Weiss, G. M., and Hirsh, H. 1998. Learning to Predict Rare Events in Event Sequences. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, AAAI Press, 359-363.