Learning to Predict Extremely Rare Events

Gary M. Weiss* and Haym Hirsh

Department of Computer Science
Rutgers University
New Brunswick, NJ 08903
gmweiss@att.com, hirsh@cs.rutgers.edu

Abstract

This paper describes Timeweaver, a genetic-based machine learning system that predicts events by identifying temporal and sequential patterns in data. This paper then focuses on the issues related to predicting *rare* events and discusses how Timeweaver addresses these issues. In particular, we describe how the genetic algorithm's fitness function is tailored to handle the prediction of rare events, by factoring in the precision and recall of each prediction rule.

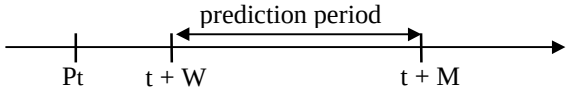
Introduction

There is currently a great deal of interest in using automated methods—particularly data mining and machine learning methods—to analyze the enormous amount of data that is routinely being collected. An important class of problems involves predicting future events based on a history of past events. A particular instance of this class of problems involves predicting hardware component failures that occur within 4ESS switches in the AT&T network, based on sequences of alarms that are reported by the 4ESS monitoring software. This task, like the majority of fault prediction tasks, involves predicting an atypical event (i.e., fault). Predicting fraudulent credit card purchases from a history of credit card transactions is another such task. Event prediction tasks that do not involve faults or failures still often involve predicting rare events—because *rare* events are often *interesting* events. For example, a stock market analyst is more likely to be interested in predicting unusual behavior—such as which stocks will double in value in the next fiscal quarter—than predicting more typical behavior, like which stocks will appreciate 10%.

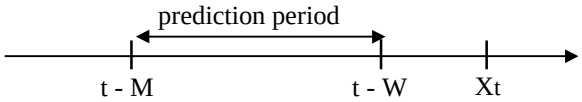
This paper begins by describing Timeweaver, a genetic-based machine learning system for predicting events, and then discusses the issues related to predicting *rare* events and how Timeweaver addresses these issues. For a more detailed description of Timeweaver, see Weiss and Hirsh (1998) or Weiss (1999).

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 event to be predicted. 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". For 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. A prediction occurring at time t, P_t, is said to be *correct* if a target event occurs within its *prediction period*. Note that we do not explicitly predict that a target event will *not* occur. 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. Thus, the monitoring time controls the specificity of the prediction (the larger 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:



Before we can attempt to solve the event prediction problem, it is necessary to have a way of evaluating a potential solution. Predictive accuracy is the most common evaluation metric for prediction (as well as classification) problems. However, because the target event is expected to occur very infrequently, accuracy is not an appropriate evaluation measure—a solution that never predicts the target event could have high accuracy. Instead, we use precision and recall to evaluate a solution. In the context of this problem, r*ecall* is the percentage of target events correctly predicted and *precision* is the percentage of times that a target event is predicted and actually occurs. By using these measures, we can focus on the performance of a potential solution at making predictions in the situation where a target event does occur. The specific way in which

*Also AT&T Labs, 20 Knightsbridge Road, Piscataway NJ 08854

the precision and recall metrics are combined to form a single evaluation function for directing the search process is described later in this paper.

The General Approach

Our approach toward solving the event prediction problem involves two steps. In the first step, a genetic algorithm (GA) 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 in the population is only part of a complete solution. A second step is used to generate a family of prediction strategies, by ordering the patterns from best to worst (based primarily on precision) and then incrementally adding one pattern at a time. The performance of the resulting family of prediction strategies is easily represented using a precision/recall curve. A user can then use this curve to help determine a good prediction strategy.

The Genetic Algorithm

The genetic algorithm is responsible for identifying patterns that predict the future occurrence of a target event. 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**

Each individual in the GA is a prediction pattern—a pattern used to predict target events. The language used to describe these patterns is straightforward and is similar to the language used to represent the raw event sequences. A prediction pattern is simply a sequence of events, but with three extensions. First, each feature value within an event may take on the wildcard value, so that it will match any feature value. Secondly, ordering constraints are specified between consecutive events, so that we can require that one event occur before another, or that the two events can occur in any order. Finally, each pattern has a pattern duration, so that the pattern can only match a sequence of events if all events involved in the match occur within a time period that does not exceed the pattern duration. 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.*

The population is initialized by creating prediction patterns containing a single event, with the feature values set 50% of the time to the wildcard value and the remaining time to a randomly selected feature value. The GA continues until either a pre-specified number of iterations are executed or the performance of the population peaks. The mutation operator randomly modifies a prediction pattern, changing the feature values, ordering primitives, and/or the pattern duration. Crossover is accomplished via a variable length crossover operator, so that the offspring of two patterns may have a different number of events than their parents. This ensures that over time prediction patterns of any size can be generated.

The Selection and Replacement Strategy

The GA's selection and replacement strategies must balance two opposing criteria: they must focus the search in the most profitable areas of the search space but also maintain a diverse population, to avoid premature convergence and to ensure that the individuals in the population collectively cover most of the target events. An additional challenge in our case is to make sure that the GA effectively handles the case where the target events occur very infrequently.

As described earlier, our evaluation function is based on both precision and recall. In a genetic algorithm the fitness function, which controls which individuals are allowed to reproduce, serves as the evaluation function. Timeweaver's fitness function combines precision and recall into a single value using the F-measure, which is used in information retrieval. The F-measure is defined below in equation 1 (Van Rijsbergen 1979).

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

The value of β, which controls the relative importance of precision to recall, is changed each iteration of the GA, so that it cycles through the values 0 to 1 using a step-size of 0.10. By varying this parameter, the GA is able to maintain patterns that are quite general as well as those that are quite specific but highly precise. The importance of this fitness function is discussed later in the next section.

To encourage diversity, we use a *niching* strategy called *sharing* that rewards individuals based on how different they are from other individuals in the population (Goldberg 1989). Individuals are selected proportional to their *shared fitness*, which is defined as fitness divided by niche count. The niche count, defined in equation 2, measures the degree of similarity of individual *i* to the *n* individuals comprising the population.

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

The similarity of two individuals is measured using a phenotypic distance metric that measures the distance based on the *performance* of the individuals at predicting the target event. Each individual prediction rule has associated with it a prediction vector, which contains one bit for each target event in the training set. Each bit is set to 1 if the rule successfully predicts the corresponding target event; otherwise it is set to 0. The distance between two individuals is simply the fraction of bit positions in the two prediction vectors that differ. 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. Note that because the target events are rare, the prediction vectors do not require much space and the number of computations required to compute the niche count is proportional to the number of target events, not to the overall dataset 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 the patterns that perform poorly on precision *and* recall are most likely to be deleted.

Results

This section shows the results of using Timeweaver to solve the telecommunication fault prediction problem, and then compares its performance to three other machine learning methods.

Results from Timeweaver

Timeweaver was applied to the problem of predicting telecommunication equipment failures from historical alarm data. The data contains 250,000 alarms reported from 75 4ESS™ switches, of which 1,200 of the alarms indicate distinct equipment failures. The training set includes 75% of the alarms and the test set the remaining 25% (from different 4ESS switches). The results shown in Figure 2 are using a 20-second warning time and an 8-hour monitoring time, and show 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 (evaluated on the training set) from the first 2000 iterations.

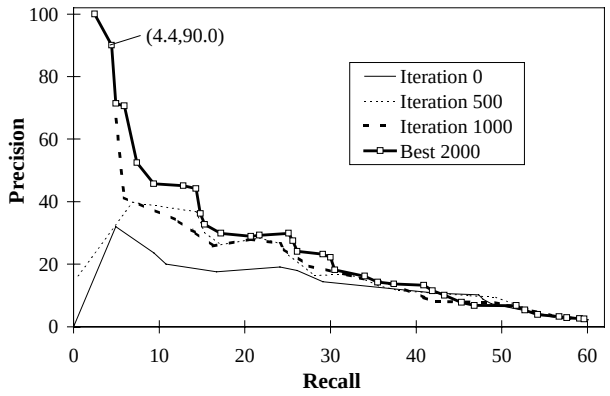


Figure 2: Learning to Predict Equipment Failures

The results shown in Figure 2 can be compared with the strategy of predicting a failure every warning time (20 seconds), which yields a precision of 3% and a recall of 63% (note that the curves generated by Timeweaver converge to this value). A recall greater than 63% is never achieved since 37% of the failures have no events within their prediction period and hence cannot be predicted within our framework.

Comparison with Other Methods

Timeweaver's performance on the equipment failure prediction problem was 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 transformed into labeled examples by sliding a window of size *n* over the data and combining the *n* events within the window into a single example by viewing the events as a single, long event (Dietterich and Michalski 1985). The example is assigned a label based on whether a target event occurs within the prediction period associated with the events within the window.

Since 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, since they are designed to optimize for predictive accuracy. 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 3. Note that in Figure 3 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.

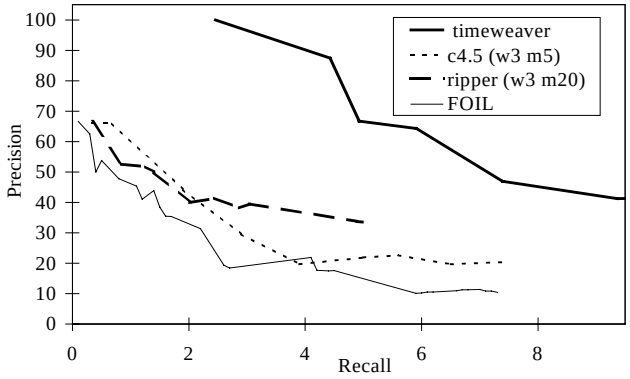


Figure 3: Comparison with Other ML Methods

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, but produced a meaningful solution without the use of anything analogous to misclassification cost. All three learning systems achieved only low levels of recall (note the limited range

on the x-axis), whereas Timeweaver is able to achieve 60% recall and superior precision. 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. Consequently, neither C4.5 nor Ripper could generate a solution with recall greater than 8%.

Discussion

In the remainder of this paper we discuss issues related to handling rare events and how Timeweaver addresses these issues. It is important to note that most of the issues we faced in predicting rare events, and the ways in which we addressed these issues, apply equally well to classification problems involving skewed class distributions.

Before we examine these issues, we would like to point out the importance of the problem formulation. The rarity of an event is based not only on the number of times it occurs, but also on the prediction period. If one event is generated every minute but a target event occurs only once a day, we might conclude the target event is rare. However, if the prediction period associated with the prediction problem is 12 hours, then given the formulation of the problem, the target event is not rare—the prediction of the target event is appropriate one-half of the time. Nonetheless, for most real-world problems, including our equipment failure prediction problem, even with this formulation most of the target events are still quite rare.

Issues with Predicting Rare Events

In this section we discuss several issues that arise when learning to predict rare events.

Issue 1: Predictive Accuracy is not Adequate. Predictive accuracy, as noted earlier, is a poor evaluation measure when we are trying to predict rare events, since we are more interested in the ability to predict when a rare event will occur, than when it will not. That is, we are willing to accept solutions with relatively low precision, as long as they predict many of the target events (i.e., have reasonable recall). One reason we are willing to accept relatively low levels of precision is that many real-world event prediction problems are extremely difficult (partly *because* the event to be predicted is rare), so that precision levels that would normally be considered to be quite poor may actually be acceptable. For example, predicting the occurrence of a fault within a 4ESS switch with 30% precision is actually quite impressive, since faults are extremely rare.

The most common methods for handling this issue involves either sampling the examples non-uniformly or using some cost-sensitive evaluation function. The sampling method involves either oversampling the minority class or undersampling the majority class. First, note that applying the sampling method to the event prediction problem is not completely straightforward, since our dataset is not made up of unordered examples. Nonetheless, this method can be applied by oversampling the subsequences of events that occur before the target events (i.e., within the prediction period associated with the target event) or undersampling the other subsequences of events. However, there are many issues involved in doing this, such as how to best divide the events into examples. It is preferable to avoid these issues entirely.

It is important to note that the undersampling method is quite disadvantageous, since the (numerous) subsequences of events not associated with a target event still serve an important function—they help us distinguish between spurious correlations in the data and patterns that are actually predictive. We are especially susceptible to the problem of finding spurious correlations given our expressive concept description language and the fact that we may only have a small number of target events (in an absolute sense). Oversampling the majority class or using a cost-sensitive evaluation function do not have this problem, but both require us to determine the most appropriate sampling ratio or cost function, which is often not known. This leads to the next issue.

Issue 2: Tuning the Solution. The user of the learning system will often not know, a priori, how to determine or identify the best prediction strategy. Consequently, most users would prefer to be given a set of solutions, such as those described by the precision/recall curve in Figure 2, and be able to choose one, rather than to specify misclassification costs or sampling rates and then accept the one solution that is then generated. In order to generate multiple solutions using these other methods, the learner would have to be run repeatedly with different parameters. We prefer a method that can generate multiple solutions in a single run.

Issue 3: Problems with Search. Many learning methods employ a greedy search in order to ensure that the learning problem is tractable. For example, a program like C4.5 evaluates the result of splitting on each feature, and then chooses the best feature to split on. For those cases where the learning problem is relatively easy, a greedy search is likely to produce results that are quite good. However, a greedy search is less appropriate for the problems we are interested in because 1) the patterns we are looking for may be a combination of several events and 2) high precision may not be achievable and 3) the target events are rare. It is the combination of these three conditions that make the problem so difficult, and make it less likely that looking at a single feature will direct the search toward the correct solution.

Another, somewhat related problem is that search methods that employ a divide-and-conquer approach often do not make the best use of limited data. To see this, imagine we have 10,000 negative examples and 100 positive examples, and a decision tree method is used to split the data. Assuming the problem is difficult, even using binary splits, after the first split there may only be 40 positive examples in one internal node and 60 in the other. Once this split occurs, the learning algorithm will have at most 60 positive examples to work with. Algorithms that do not use a

divide-and-conquer approach are not subject to this specific problem.

Using a GA to Predict Rare Events

We chose to use a genetic algorithm to predict rare events because it successfully addresses the three issues described in the previous section. Specifically, it permits us to use an evaluation function based on precision and recall, and, by allowing us simultaneously evolve prediction rules that are optimized for different weightings of precision and recall, allowed us to generate a family of solutions within one run. Furthermore, a genetic algorithm is a powerful search method that is not greedy (it is good at finding optimal values for multiple features) and does not employ a divide-and-conquer approach. However, we need to acknowledge that our decision was also influenced by the fact that existing classification systems were not readily applicable. If it were possible to *effectively* represent the prediction problem as a classification problem, or if there were prediction methods that could handle the notion of a prediction period, we might have tried to adapt existing methods by employing a sampling strategy or by using misclassification costs.

The single most important decision in our GA was the choice of the fitness function, which is responsible for directing the search process. Once the decision was made to base the fitness function on precision and recall, various schemes of weighting the two were tried. Some of these schemes weighted the two factors in a static manner (e.g., precision was valued twice as much as recall). None of these schemes performed well, no matter what the relative weighting of the two components. In virtually all cases the GA quickly converged to (and got stuck at) one of two extremes: a population with many precise, highly specific patterns with low recall, or a population with many general patterns with low precision. It appears that to make the most progress, the population quickly moves to one of these extremes, and once there it is difficult to make a change that leads to a better prediction pattern. We also tried an approach reminiscent of simulated annealing by beginning with a fitness function that placed more emphasis on recall, and gradually increased the relative importance of precision. This approach suffered from the same problem as the approach using a static weighting of precision and recall.

We finally tried a fitness function based on the F-measure, as defined previously in equation 1. This did not yield significantly better results, until we varied the relative importance of precision and recall each cycle of the GA. This ensured that a diverse set of patterns were maintained—some which were general, some which were specific, and some which were in between. This strategy led to good results and also allowed us to generate a family of solutions (by combining different numbers of patterns) which spanned a wide range of recall values. We believe this way of searching the search space was essential in learning to predict the rare events.

Conclusion

In this paper we described how a genetic algorithm, with an appropriately engineered fitness function, can identify rules for predicting extremely rare events. We discussed advantages of this approach and why we feel it is appropriate for many real-world problems involving unbalanced datasets. We would like to point out that having the ability to specify the evaluation function for a learning system provides a lot of flexibility, and enabled us not only to handle the unbalanced data problem, but also to handle other aspects of the problem (Weiss and Hirsh 1998). We believe that the key issue in designing our genetic algorithm was to develop a fitness function that appropriately factored in precision and recall, so that extremely rare events could be predicted.

References

Cohen, W. 1995. Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, 115-123.

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

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

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.

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

Weiss, G. M. 1999. Timeweaver: A Genetic Algorithm for Identifying Predictive Patterns in Sequences of Events. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 718-725. San Francisco, Calif.: Morgan Kaufmann

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*, 359-363. Menlo Park, Calif: AAAI Press.