To appear in the journal *Data Mining and Knowledge Discovery*

Discovering Interesting Patterns for Investment Decision Making with GLOWER ☹ - A Genetic Learner Overlaid With Entropy Reduction

Vasant Dhar[1]
Dashin Chou
Foster Provost

Working Paper # CIIO 00-02
Center for Information Intensive Organizations

Stern School of Business
New York University
44 West 4th Street, Room 9-75
New York NY 10012

January 2000

Abstract

Prediction in financial domains is notoriously difficult for a number of reasons. First, theories tend to be weak or non-existent, which makes problem formulation open ended by forcing us to consider a large number of independent variables and thereby increasing the dimensionality of the search space. Second, the weak relationships among variables tend to be nonlinear, and may hold only in limited areas of the search space. Third, in financial practice, where analysts conduct extensive manual analysis of historically well performing indicators, a key is to find the hidden interactions among variables that perform well in combination. Unfortunately, these are exactly the patterns that the greedy search biases incorporated by many standard rule learning algorithms will miss. In this paper, we describe and evaluate several variations of a new genetic learning algorithm (GLOWER) on a variety of data sets. The design of GLOWER has been motivated by financial prediction problems, but incorporates successful ideas from tree induction and rule learning. We examine the performance of several GLOWER variants on two UCI data sets as well as on a standard financial prediction problem (S&P500 stock returns), using the results to identify one of the better variants for further comparisons. We introduce a new (to KDD) financial prediction problem (predicting positive and negative earnings surprises), and experiment with GLOWER, contrasting it with tree- and rule-induction approaches. Our results are encouraging, showing that GLOWER has the ability to uncover effective patterns for difficult problems that have weak structure and significant nonlinearities.

Keywords: Data Mining, Knowledge Discovery, Machine Learning, Genetic Algorithms, Financial Prediction, Rule Learning, Investment Decision Making, Systematic Trading

[1] Email: vdhar@stern.nyu.edu FAX: (212) 995-4228 TEL: (212) 998-0816

1. Introduction

Our experience in financial domains is that decision makers are more likely to invest capital using models that are easy to understand. More specifically, decision makers want to understand *when* to pay attention to specific market indicators, and in particular, in what ranges and under what conditions these indicators produce good risk-adjusted returns. Indeed, many professional traders have remarked that they are occasionally inclined to make predictions about market volatility and direction, but cannot specify these conditions precisely or with any degree of confidence. Rules generated by pattern discovery algorithms are particularly appealing in this respect because they can make explicit to the decision maker the particular interactions among the various market indicators that produce desirable results. They can offer the decision maker a "loose theory" about the problem that is easy to critique.

Financial prediction problems tend to be very difficult to model. Investment professionals who use systematic trading strategies invariably experience periods where their models fail. Modelers often refer to these periods as "noise," although it can be argued that the so-called noise arises from the limitations of the model rather than from unpredictable aspects of the problem.

What are the characteristics of financial problems that make it difficult to induce robust predictive models? First, the dimensionality of the problem is high. It is common practice, for example, to derive "telescoped" moving averages of variables (Barr and Mani, 1994) in order to be able to capture the impact of temporal histories of different lengths, such as 10, 60, 250 prior data points (days, minutes, etc). This enables the discovery of patterns that capture not only long- and short-term relationships, but also the transitions between them. For example, if a long-term indicator is high but a short-term indicator is low, it may indicate a recent "cooling down" phenomenon. While the use of telescoping allows for the discovery of such effects, it increases the dimensionality, and correspondingly, the size of the search space increases exponentially.

Secondly, relationships among independent and dependent variables are weak and nonlinear. The nonlinearities can be especially pronounced towards the tails of distributions, where a correlation becomes stronger or weaker than it is elsewhere. For example, a common type of nonlinearity in technical analysis is trend reversal, where price trends change direction after a prolonged period. In this case, "more" (trend) is not always better; the hazard of assuming that the trend will continue may increase as the trend continues. Similarly, an earnings revision on a stock by an analyst may have no impact on its price *unless* the revision exceeds some threshold. In other words, the effect holds only in the tail-end of the distribution.

Thirdly, variable *interactions* can be significant. For example, we may observe that a "negative earnings surprise" (i.e., the earnings reported by a company are lower than expected) has *no effect* on returns in general. On the other hand, we may find that if we were to make the rule more specific, by eliminating "market leaders" in the "technology"

sector, the effect is dramatic. It is important for a learning algorithm to be able to discover such interaction effects in a way that makes the induced relationship as accurate and general as possible. In domains such as these, where much manual analysis concentrates on following trails of well-performing indicators, it is exactly the hidden interactions that are important.

Our basic assumption in predicting financial markets is that it is not possible to do so *most* of the time. This is consistent with remarks many financial professionals have made. In particular, many trading professionals do feel that there are times, admittedly few, where they can predict relatively well. This philosophy, "generally agnostic but occasionally making bets," has important implications for how we approach the modeling problem. One of the major challenges is to reduce the "noisy" periods by being more selective about the conditions under which to invest--to find patterns that offer a reasonable number of opportunities to conduct high risk-adjusted-return trades. In doing so, we must consider explicitly the tradeoff between model coverage and model accuracy. Trying to give an accurate prediction for all data points is unlikely to succeed. On the other hand, a single small, accurate rule probably will not apply often enough to be worthwhile. The model (for us a set of rules) must be accurate enough and simultaneously general enough to allow sufficient high-probability opportunities to trade effectively.

In the next section we go into more detail on the benefits and limitations of genetic search for data mining. It turns out that we can address the limitations by enhancing genetic search with basic heuristics inspired by work on tree induction and rule induction. We present results comparing the augmented genetic learning algorithm (GLOWER) to tree induction and rule induction for a difficult financial prediction problem. The results are as we would expect based on an analysis of the strengths and weaknesses of the various approaches: GLOWER is able to find significantly better rules than its competitors based on the domain-specific notion of rule quality.

In section 3 we define the problem-specific notions of accuracy and generality in terms of confidence and support. In section 4 we introduce a "generic" algorithm used for genetic rule discovery, its dynamics, and extensions that take advantage of niching techniques that are commonly employed by genetic algorithms. In section 5 we introduce a new hybrid genetic rule-learning algorithm that combines an entropy reduction heuristic (as used by TI algorithms) and an inductive strengthening heuristic (as used by rule induction algorithms) with standard genetic search. We describe how we used three benchmark data sets to do a comparison of several options for instantiating the genetic algorithm. After choosing a particular instantiation, based on the results of this study, in section 7 we apply GLOWER to a new (to KDD) financial problem, *predicting earnings surprises*, and contrast the results with those obtained with tree induction and with rule learning. Finally, we discuss further considerations in using genetic algorithms for pattern discovery in finance, and further directions for research.

2. Benefits and Limitations of Genetic Search for Rules

Our objective is to find rule-like patterns in the data. Various candidate algorithms for doing so have been studied, most notably tree induction (TI) algorithms (Quinlan, 1986; Breiman et al. 1984), separate-and-conquer rule-learning algorithms (Furnkranz, 1999), and systematic rule-space search algorithms (Provost, Aronis and Buchanan, 1999). These algorithms all search the space of conjunctive rules. Most search the space from general rules (syntactically simple, covering many data) to specific rules (having more conditions, covering fewer data).

The KDD literature has paid less attention to genetic algorithms for searching the rule space. Genetic algorithms (Packard, 1989; Goldberg, 1989, Holland, 1992) have been shown to be well suited to learning rule-like patterns. They have the ability to search large spaces for patterns without resorting to heuristics that are biased against term interactions. In this paper, we focus on the use of genetic algorithms—particularly as applied to financial data mining problems. To provide contrast with common data mining practice, we pay particular attention to how genetic algorithms differ from tree and rule induction algorithms.

Genetic algorithms have several advantages as a rule discovery method. Their two primary advantages are the ability to scour a search space thoroughly, and the ability to allow arbitrary fitness functions in the search. Their main disadvantages are lack of speed, randomness in creating the initial population (and in exploration as well, although some may consider this an advantage), and the fact that they can be myopic after they find one good solution. We address the limitations except for speed, which is beyond the scope of this article, and demonstrate the first two benefits.

2.1 Limitations of Genetic Search

Traditional genetic algorithms have several limitations when used for rule mining. One drawback is the random creation of initial populations and the randomness of subsequent exploration. Why should we start with a random population? Although GLOWER retains the positive aspects of randomness such as the ability to escape local maxima, we also overlay entropy reduction to focus the genetic search, both to build the initial population and in subsequent exploration.

A second important limitation of genetic algorithms for rule mining is that they have a tendency to focus too closely on a single, high-quality solution. The "building blocks" (Holland, 1975) of this single solution can distribute themselves rapidly throughout the population, and tend to elbow out other potentially good solutions. To address this problem, we evaluate several refocusing methods from the literature on genetic algorithms and from the literature on rule learning.

A third limitation of genetic search is it is comparatively slow, because it re-evaluates large populations of rules over and over after making small changes. GLOWER's implementation is highly optimized in terms of its internal representation for fast query and evaluation, in the spirit of the RETE algorithm (Forgy, 1982). However, run time and implementation issues are beyond the scope of this paper.

We want to stress that we are not claiming that genetic algorithms are generally better than the other search techniques for finding interesting, useful rules. We do believe that they are a useful alternative, with attractive properties. They should receive greater attention in the KDD literature, especially for noisy domains like finance where it is important to find small patterns based on combinations of conditions including numeric variables. We also want to note that there has been a large volume of work on genetic algorithms, both theoretical and empirical. This paper is not meant as a survey of that field; interested readers should consider the brief overview recently provided by DeJong (1999), and work by Goldberg (1989), Packard (1989), Holland (1995) and others.

In order to appreciate the "fixes" that are necessary to the genetic algorithm for rule learning, it is appropriate to begin by considering the basic limitations of greedy search heuristics used in machine learning algorithms.

2.2 Benefits of Genetic Search

Figure 1 helps to locate genetic algorithms on the rule-mining landscape. It depicts a spectrum of search techniques in terms of the thoroughness of search that they perform. On one end of the spectrum are tree induction algorithms that use a highly greedy heuristic and perform an irrevocable search. Conventional rule learning algorithms consider a wider variety of alternatives and therefore are more thorough.[2] At the other end of the spectrum, genetic algorithms are capable of conducting very thorough searches of the space because they are less restricted by a greedy search bias. Genetic search performs implicit backtracking in its search of the rule space, thereby allowing it to find complex interactions that the other non-backtracking searches would miss. To illustrate this property, which we believe to be important in financial domains, we treat in detail below the problem with using greedy search for rule learning.

[2] These consist of "separate and conquer" type algorthms (Fuhrenkranz, 1999) and "systematic rule-based search type" algorithms (Provost et.al, 1999). If only categorical attributes are considered, systematic rule learning algorithms perform *very* thorough searches of the rule space. However, for this paper we are interested in domains that include (and in fact comprise primarily) continuous attributes, for which systematic rule-space search algorithms are less thorough.

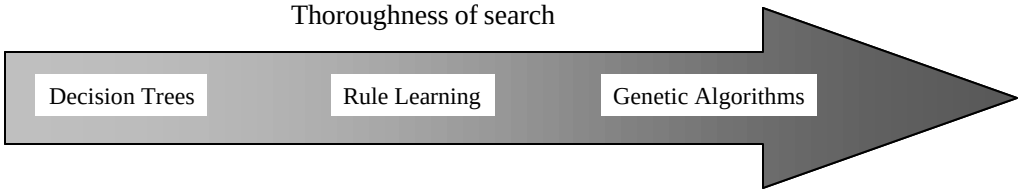

Figure 1

GAs have the additional advantage, over other conventional rule-learning algorithms, of comparing among a set of competing candidate rules as search is conducted. Tree induction algorithms evaluate splits locally, comparing few rules, and doing so only implicitly. Other rule-learning algorithms compare rules to fixed or user-specified criteria, but rarely against each other during the search.[3] A defining characteristic of genetic search is that rules compete against each other, based on some fitness criterion. This is especially useful in domains where the target evaluation function is not well specified at the outset. Unlike many rule-learning algorithms, which are fine-tuned for a particular evaluation function (e.g., for maximal classification accuracy), genetic rule-learning algorithms can accept arbitrary evaluation criteria as input, including the ability to penalize overlap among rules. We will see later that this allows us to find small sets of rules that score well with respect to a problem-specific quality measure, dealing explicitly with the commonly noted problem of finding "too many" rules, including many small variants of some core pattern.

2.3. The failings of greedy search

Tree Induction algorithms are currently among the most widely used techniques in data mining. They are fast, are surprisingly effective at finding accurate classifiers with little knob twiddling, and produce explicit decision trees from which rules can be extracted. Another strength of TI algorithms is that they classify the data *completely*. Every datum can be classified into a particular derived class, resulting in 100% coverage of the data.

But tree induction algorithms generally trade off some accuracy for speed. Most TI techniques use *recursive partitioning*: choose a node in the tree, and evaluate competing univariate splits on the original data set based on their ability to produce statistical differences in the distribution of the dependent variable. However, regardless of how judiciously the algorithm splits the data, the greedy heuristic may overlook multivariate relationships that are not apparent by viewing individual variables in isolation.

The following example illustrates how the greedy search conducted by TI algorithms can overlook good patterns. It also illustrates how these techniques can be limited in their

[3] We should note that separate-and-conquer rule-learning algorithms implicitly do a limited form of comparison of hypothesized rules during the search—not enough to warrant a comprehensive discussion here. The "conquer without separating" rule-learning algorithm CWS (Domingos, 1996b) compares rules explicitly as they are learned.

ability to handle nonlinearities such as interaction effects among problem variables. Consider the simple example database shown in Table 1, which comprises 20 records with the attributes:

`Gender, Age, State, Consumption`

`Consumption` represents the dependent variable. We would like to use this database to build a model that will predict `Consumption` for previously unseen records. In this simple example, `Consumption` is the total dollar amount spent by an individual on roller-blades during a selected time period, and is coded as "`High`" or "`Low`" based upon problem-specific criteria. `State` and `Gender` are categorical variables. `Age` is a continuous variable.

Gender	Age	State	Consumption
F	36	CA	High
F	36	CA	High
F	36	NY	High
F	36	CA	Low
F	36	CA	Low
F	34	CA	Low
F	34	CA	Low
M	34	CA	High
M	34	CA	High
M	34	CA	High
M	34	CA	High
M	34	CA	High
M	34	NY	Low
M	34	NY	Low
M	34	NY	Low
M	36	NY	High
M	36	NY	High
M	36	CA	Low
M	36	CA	Low
M	36	CA	Low

Table 1: A small data set

Figure 2 shows how a tree induction algorithm, CART (Breiman et al., 1984), classifies the above data (restricting splits to nodes with at least 10 cases). The leftmost cluster in Figure 1 shows the complete data set, containing 10 `High` and 10 `Low` consumers as circles and crosses, respectively.

The first split, on `Gender`, produces a slightly higher proportion of "`High`" consumers. In fact, it is the *only* attribute on which a split produces any improvement at all using the greedy splitting heuristic. The `Male` group is further partitioned on `Age`, under and over 35, yielding a cluster where 62.5% of the cases have `Consumption = High`. The "rule" or pattern that emerges is that males under the age of 35 belong to the `High Consumption` category:

```
IF Gender = ″Male″ AND Age < 35 THEN Consumption = ″High″    (Rule 1)
```

The parts of the rule before and after the "THEN" are referred to as the antecedent and the consequent of the rule, respectively.

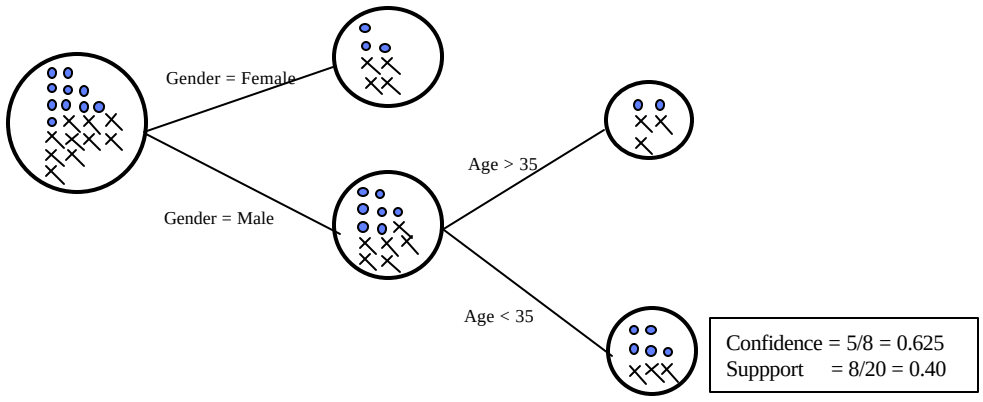

Figure 2: An example of how a Tree Induction algorithm partitions data into homogeneous classes (cross = class 1 = "low" consumers, circle = class 2 = "high" consumers)

In the example above, each split is on a single variable. Tree induction algorithms typically determine split points based on a heuristic such as entropy reduction (which, as described below, we use to augment a more traditional genetic algorithm). For example, the entropy of a cluster *i* can be computed using the standard formula:

$$H_i = -\sum_k p_k \log_2(p_k) \qquad (1)$$

where p_k is the probability that an example picked at random from the cluster *i* belongs to the k^{th} class. When applied to a cluster *i*, H_i, the entropy, measures the average amount (number of bits) of information required to identify the class of an example in the cluster. The entropy of a cluster is minimum where the probability is 1; that is, all members belong to the same class. Entropy is maximum where an example is equally likely to belong to any class, as in the leftmost cluster of Figure 2, where the probability that an example picked at random will belong to either class is identical, in this case, 0.5.

The *gain* from a split is computed based on the difference between the entropy of the parent cluster and the entropy of the child clusters resulting from the split. That is, if a cluster *i* is partitioned into *j* subsets:

$$gain_{ij} = \mathrm{H}_i - \sum_j H_j * R_j \qquad (2)$$

where R_j is the ratio of the number of cases in cluster *j* to those in *i*. This is an information-theoretic measure of the *value* of information obtained from the split. It is the amount of discrimination that the split produces on the distribution of the dependent variable. TI algorithms use a measure such as information gain to compare the potential splits that can be placed at each node in the decision tree (using, typically, a depth-first search). The split with the best score is placed at the node; the (sub)set of data at that

node is further partitioned based on the split, and the procedure is applied recursively to each subset until some stopping criteria are met.[4]

This greedy search is the reason why TI algorithms are fast (Lim, Loh and Shih (2000) show just how fast they are). The computation of the splitting metric is simple, and there is no backtracking. This enables the algorithm, in many cases, to process databases with hundreds of thousands of records in seconds on a powerful workstation.

But why does such an algorithm overlook "good" relationships in the data? Recall that the split on `Gender` was the only split that produced an improvement. In fact, if we had split on `State`, yielding *no* immediately apparent improvement, and then again on `Age`, we would have obtained a better rule than did the greedy search—a rule with higher confidence, and comparable support. This is shown in Figure 3. The TI algorithm did not discover this pattern because the split on `State` does not produce improved clusters (the older people in California are not heavy consumers); it does not reduce the entropy of the original data. The algorithm has no way to recover from its initial and irrevocable greedy split. We believe that in domains with weak structure such as in finance, there are many valuable patterns that TI algorithms overlook for the reasons described above.

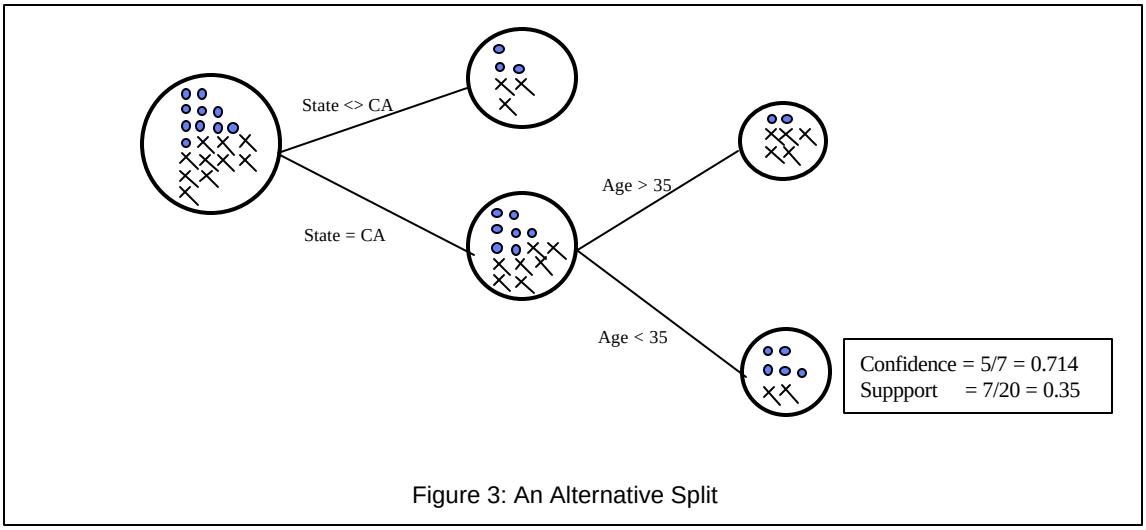

Figure 3: An Alternative Split

Rule learning algorithms are further toward the thorough end of the spectrum in Figure 1, because their search is "less greedy." Specifically, these algorithms typically consider several independent paths in the search space--all need not be rooted at the same node (Clark and Niblett, 1989; Clearwater and Provost, 1990; Furnkranz, 1999; Hong, 1991; Smyth and Goodman 1991; Provost, Aronis and Buchanan, 1999). They are better equipped to find the smaller patterns that the irrevocable search strategy of tree induction misses.

[4] In computing the goodness of a split, this *gain* value needs to be normalized so that fewer splits and larger clusters are favored over smaller ones. Otherwise the algorithm would be overly biased towards producing very small clusters (in the extreme case, of size 1 since these would minimize entropy). There are a number of heuristics for implementing this normalization (see, for example, Quinlan (1993), Breiman *et al.* (1984), or Michie, *et al.* (1994)).

It is the search for small, non-overlapping, and useful patterns including continuous variables that concerns us. Our experience is that predictability in financial domains is highly elusive, and is possible only infrequently. We have found that minor changes in discretization intervals and granularity can cause the search to produce significantly different outputs. A solution to these problems is to perform more search, and at the same time to be more selective. We have found genetic search to be particularly effective in this respect. However, we believe that techniques such as entropy minimization are conceptually sound and are quite useful. Rather than discarding them, it is worthwhile for the genetic search to incorporate them.

3. Evaluation of partial models

Evaluation of models typically comes in two flavors. Many systems, such as the TI algorithms described above, produce models that are intended to apply to all the data (100% coverage). Such models often are evaluated by the expected number of errors that they would make on (previously unseen) examples drawn from some distribution. The other flavor of evaluation is to look at individual rules (or other small-coverage patterns) and evaluate them outside the context of the model that together they would form.

Two commonly used metrics used to measure the goodness of individual rules are *confidence* and *support*. These metrics have been used for many years in a variety of rule-learning algorithms, and have become especially popular in the KDD community because of their use in association-rule algorithms. *Confidence* measures the correlation between the antecedent and the consequent of the rule. *Support* measures how much of the data set the rule covers. If N is the total number of examples in a data set, then, for a rule of the form $A \rightarrow B$:

confidence = (Number of cases satisfying A and B) / (Number of cases satisfying A)
= $p(A \cap B) / p(A)$

support = (Number of cases satisfying A) / N
= $p(A)$

error rate = 1 - confidence

For example, the confidence of Rule 1 from Section 1 is 0.625 (5/8), whereas the support is 0.40 (8/20).

These two flavors of evaluation actually apply similar measures to two ends of a spectrum of partial models. On one extreme we could consider 100% support, and report the error rate (1-confidence) associated with full coverage of the data. On the other extreme, w could focus on partial models at the finest granularity, reporting the error rate of an individual rule and its associated support. We are interested in partial models along the spectrum between these two extremes, in balancing confidence and support.

For example, consider the financial problem of predicting earnings surprises. The task is difficult, and it is unlikely that a model could be built that classifies all cases accurately. On the other hand, viewing the statistics of individual rules out of the context of use (presumably as part of a larger model) provides less insight than we would like. Our goal is to find models, perhaps comprising a few rules, that predict a useful number of earnings surprises with high accuracy. Of course, the definitions of "useful" and "high" are problem dependent, to which we will return when we discuss further the earnings-surprise prediction problem. Fortunately, statistics such as confidence and support apply not only to the two common flavors, but are well suited across the entire spectrum of partial models. In fact, if so inclined, one could graph the tradeoff between the two as partial models are constructed rule by rule.

4. Genetic Rule Discovery

In financial domains, decision makers often are interested in finding easy-to-understand rules to govern investment performance. For example, a qualitative rule might be "if the *short-term* moving average of prices exceeds the *long-term* moving average, and the trading volume over the long term is *increasing*, buy." In this case, a discovery algorithm could fill in the blanks denoted by the italicized phrases. For example, it might find that the best *buying* results (i.e. "going long") occur when the following rule is applied:

Short_term = 5 days,
Long_term = 30 days,
Short_term_moving_average > *Long_term_moving_average*,
Long_term_volume_rate > 2 percent AND < 5 percent,
Holding_period = 2 days.

As we can see, the search space of possible rules is extremely large even for this trivial example with only a few variables. It should also be apparent that the representation used by the discovery algorithm must be able to deal easily with *inequality* conditions such as "at least 2 percent," "between 2 and 5 percent," "less than 2 or greater than 10," and so on. In order to make the collection of rules more easily understandable as a whole, it is generally desirable that other *buying* rules overlap as little as possible with the one above.

4.1. Representation: Gene and Chromosome Semantics

Table 2 shows the representation of patterns used by the genetic learning algorithm. Each pattern, a *chromosome*, represents a specific rule. The genetic algorithm works with multiple hypothesized rules, which make up its *population*. A population at any point in the search is a snapshot of the solutions the algorithm is considering. The algorithm iterates, modifying its population with each *generation*. Each pattern (chromosome) is an expression defined over the problem variables and constants using the relational operators *equal*, *greater-than*, and *less-than*, and the Boolean operators *and*, *or*, and *not*. At the implementation level, chromosomes are queries issued to a database. Chromosomes in turn are composed of constraints defined over individual problem

variables. These are represented as sets of *genes*. At the lowest level, a gene represents the smallest element of a constraint, namely, a variable, constant, or an operator.

Concept	**Representation**
Variable|Constant|Operator Example: *30-day-moving average of price (MA_{30})*	Gene
Univariate predicate (Single "conjunct") Example: $MA_{30} > 10$ Example: $MA_{30} < 10$ *OR* $MA_{30} > 90$	Set of Genes
Multivariate predicate (Conjunctive pattern) Example: $MA_{30} > 10$ *AND* $MA_{10} < 5$	Chromosome
Multiple Patterns	Population

Table 2: The Concept Class Representation

The above representation is equivalent to that of tree induction algorithms such as CART in that a chromosome (rule) is equivalent to a path from a root to a terminal node in a decision tree. In addition, however, a constraint on a single variable can be a disjunct, such as "$MA_{30} < 10$ *OR* $MA_{30} > 90$". It should be noted that our system *does not* represent knowledge in the manner commonly associated with *genetic classifier systems* (Goldberg, 1989; Holland, 1992) where individual chromosomes may represent sub-components or interim results that make up some larger chain of reasoning represented by groups of chromosomes. We view genetic search simply as an alternative algorithm for searching the rule space--one with particular, attractive properties.

For determining fitness, chromosomes can be evaluated based on criteria such as entropy, support, confidence, or some combination of such metrics. By controlling the numbers of constrained variables in chromosomes, genetic search can be biased to look for patterns of a desired level of specificity. This is a parameter that we can manipulate to control (to some extent) the degree of variable interactions, or nonlinearity, we want the algorithm to be capable of discovering from the data.

4.2. Schema Theory: The Basis for Genetic Rule Discovery

Holland used the term *schema* in the context of genetic algorithms to explain the theoretic basis for genetic search. His basic reasoning is that a single chromosome can implicitly cover a large part of the search space, and that a collection of them can scour a large search space thoroughly. To see why, consider a problem with, say, 30 variables. Suppose that the search is considering a hypothesis where only one variable, say *age*, is constrained, "*age < 35*," whereas all other variables are unrestricted, that is, we "don't care" about what values they take. Such a chromosome represents a *region* in the search space. Holland referred to such a region as a schema. Fewer, or more precisely, looser restrictions represent more general schemata, larger areas of the search space. For example, the constraint on age, *"age < 35"*, with "don't care" for all other variables, represents a very general schema. The constraint *"age < 35 AND state = CA"* represents

a more specific schema. These constraints, such as "*age < 35*", are referred to as *building blocks.* Holland demonstrated that if a schema happened to result in better solutions than the population average, then this schema would manifest itself through its building blocks in above average proportions (i.e., in many chromosomes) in populations of subsequent generations.

A basic feature of our representation is that it manipulates schemata directly by allowing "don't cares" for variables. By specifying the number of don't cares allowed in a chromosome, we are able to control to some extent the generality of patterns we are interested in discovering. This is an important practical consideration. It influences how easy it will be for users to interpret the discovered patterns and has a direct impact on the support and confidence of the discovered patterns. We refer to the number of specified or constrained variables in a schema as its *order*. The order of the schema corresponding to *"age < 35 AND state = CA"* therefore is 2. This corresponds directly to restricting tree depth in TI algorithms, or restricting rule complexity (or length) in other rule-space search algorithms.

4.3. Population Dynamics

The pattern discovery process with genetic search works as follows: an initial population of patterns (chromosomes) first is created, randomly or biased in some way. Each chromosome is evaluated and ranked. The higher-ranked chromosomes are selected to participate in "mating" and "mutation" to produce new offspring. Mating essentially involves exchanging parts of chromosomes (genes) between pairs. This is called *crossover*. Mutating involves changing the value of a gene, such as a number or an operator. By repeating these steps over and over, usually hundreds or thousands of times, the search converges on populations of better patterns, as measured using some fitness function. The dynamics of the search is described by the following equations.

Let $s(S,t)$ be the number of patterns corresponding to some schema S at generation t. If the fitness of schema S is f(S), and the average fitness of the population at generation t is $f(P,t)$, then the expected number of chromosomes corresponding to S in the next generation is:

$$s(S,t+1) = s(S,t)*f(S)/f(P,t) \qquad (3)$$

For simplicity, if we assume that $f(S)/f(P,t)$ is a constant, $1+c$, then

$$s(S,t+1) = s(S,0)*(1+c)^{t+1} \qquad (4)$$

Equation 4 states that selection (or reproduction) allocates members of a schema, i.e. patterns corresponding to it, at an exponentially increasing rate if their fitness is above the population average, and at an exponentially decreasing rate if below average. For example, if the average fitness of the population is 0.5 and that of the schema "age < 35" is 0.6, then 1+c would be 1.2, meaning that we would expect to see 20% additional representatives of the schema in the next generation.

The effect of crossover is to break up schemata, in particular, those that have more variables specified. The term "order" is used to designate the number of specified variables in a schema. Assuming that our crossover involves exchanging exactly *one* variable between chromosomes (regardless of position in the chromosome),[5] and ignoring cases where the chosen position is instantiated identically in the chosen chromosomes, the probability of a schema getting disrupted is $o(S)/l$ where $o(S)$ is the order of the schema and l is the chromosome length. For example, with a chromosome of length 4 and a schema of order 2, the probability of disruption would be 2/4, whereas if all are specified, the probability of disruption is 1. If we weaken our assumption and let crossover involve the exchange of m variables, the probability of survival becomes:

$$^{(l-o)}C_m \,/\, ^{l}C_m$$

If m is picked randomly, i.e. with probability $1/l$, the probability of survival is:

$$\frac{1}{l}\sum_{m=1}^{l} {}^{(l-o)}C_m \Big/ {}^{l}C_m$$

Combining the above expression with equation 4 gives the combined effect of selection and crossover:

$$s(S,t+1) = s(S,0).(1+c)^{t+1}.(1/l.\Sigma\, ^{(l-o)}C_m \,/\, ^{l}C_m) \qquad (5)$$

Finally, mutation involves changing a single value in a chromosome. If the probability of carrying out a mutation is p_m, the probability of survival through mutation is $(1 - p_m)^{o(S)}$. Since p_m is usually small, i.e. << 1, this can be approximated as $(1 - o(S).p_m)$.

The combined effect of the genetic operations of selection, crossover, and mutation is given by the following equation:

$$s(S,t+1) = s(S,0).(1+c)^{t+1}.(1/l.\Sigma\, ^{(l-o)}C_m \,/\, ^{l}C_m).(1 - o(S).p_m) \qquad (6)$$

Equation 6 expresses what is referred to in the literature as the *Schema Theorem*. It shows the competing forces on schema survival. The first part of equation 6 says that above average schemata are represented in exponentially increasing proportions in subsequent generations. The second and third parts of equation 6 say that low-order schemata have a higher chance of survival than high-order schemata.

Why is this interesting? For problems with weak structure, the low-order schemata that have significantly higher fitness than the population average are likely to be those that impose tighter bounds on the specified variables. These low-support but high-confidence patterns can serve as the seeds for higher-support patterns. The genetic algorithm thereby

[5] This is not a cut-point crossover; it is the simple exchange of a gene.

learns through seeding, that is, finding low-coverage but high-fitness patterns that can then be "expanded" until their performance begins to degrade.

We would like to find patterns with higher support, because although we are happy for our partial models to contain multiple rules, for domain-specific reasons a few high-support rules is much preferable to a large number of low-support rules. In what follows, we present and evaluate empirically a number of competing heuristics that conduct intelligent "adaptive sampling" of the database during the search, to help focus it on higher-support patterns.

4.4 Focusing Genetic Rule Discovery

In order to focus the genetic search to find interesting patterns, the algorithm "tunes" how fitness is computed depending on what already has been discovered or explored. Conceptually, we want to allow the fitness calculation (cf., equation 6) to consider the "state" of the search. Certain general focusing heuristics have been successful across other types of rule learning, and we incorporate specific instances into our genetic search.

Consider a data set with 2 independent variables, V1 and V2, and one dependent variable as shown in Figure 4. The dependent variable can take on four values, A, B, C and D. The patterns A_1 and A_2 show two disjoint clusters of the class A, corresponding to two different combinations (ranges) of independent variables. The size of each cluster denotes its importance, measured in terms of some function of confidence and support. Similarly for B and C. The remaining area is labeled D.

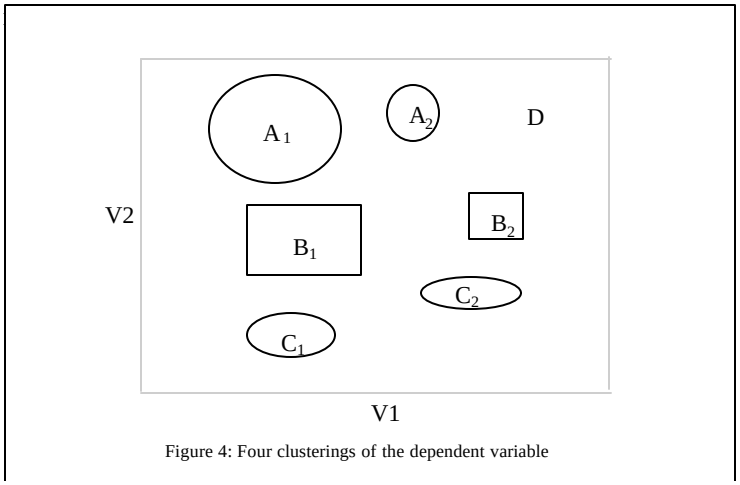

Figure 4: Four clusterings of the dependent variable

A traditional genetic rule discovery algorithm tends to do the following: all other things being equal, since the pattern A_1 is the dominant pattern in the data set, chromosomes typically gather in the neighborhood of A_1. If such a pattern dominates the early populations, the algorithm is likely to overlook the other patterns.

This example highlights two problems we would like to overcome. First, we would like to enable the genetic rule discovery algorithm to find all salient patterns, rather than

letting a few high-quality patterns dominate. Second, we would like to make each of the blobs in Figure 4 as large as possible by balancing confidence and support during search.

4.4.1 Sequential Niching: SN

One standard approach to guiding a genetic algorithm to find all solutions is to use "niching," where chromosomes are grouped into sub-populations, one for each class. Niching schemes have been described and their dynamics analyzed extensively as effective general purpose strategies for dealing with certain types of multimodal problems (Goldberg and Richardson (1987), Deb and Goldberg (1989), Oei *et al*. (1991), Goldberg *et al*. (1992)). Mahfoud (1995) also provides an extensive survey and discussion. Niching can be particularly effective when each niche can be made to focus on a particular cluster.

An alternative to requiring such a priori knowledge is to allow the algorithm to determine appropriate clusters empirically, focusing on particular parts of the space while they appear fruitful, and once they produce what seems to be a good rule, focusing the search elsewhere. Using previously learned knowledge to restrict the search to other parts of the space is a common heuristic in rule learning. The general notion has been called *inductive strengthening* (Provost & Buchanan, 1992): placing stronger restrictions on the search based on the rules that have already been induced.

Inductive strengthening is a method for adaptively adjusting an algorithm's inductive bias (Mitchell 1980).[6] Inductive bias refers to any criteria other than strict consistency with the training data that are used to select a model. Restriction bias refers to algorithm or problem design choices that restrict certain models from being chosen. A *strong* bias is very restrictive, ruling out many possible hypotheses, and correspondingly, a weak bias allows a much larger space of hypotheses to be considered. The tradeoff is that a strong bias is preferable for efficiency reasons (among others); however, unless it is well chosen, a strong bias may mask desirable rules. Algorithms incorporate inductive strengthening heuristics in an attempt to get the best of both worlds: start with a weak bias, and as good rules are induced, restrict the search to look elsewhere for additional rules.

Genetic search can perform inductive strengthening by niching *sequentially.* After the search converges on a high-fitness schema, the evaluation function can be modified to penalize patterns corresponding to it. Figure 5 shows how this works: the classified area, A_1, is marked. This change forces additional chromosomes that represent A_1 no longer to have good fitness values. Therefore, the genetic algorithm must search for other patterns, such as A_2 and B_1. Sequential niching increases the chances of finding more patterns and of increasing overall coverage.

Beasley, Bull and Martin (1993) demonstrate such a sequential niching (SN) method. It works by iterating a simple GA, and maintaining the best solution of each run off line.

[6] Provost and Buchanan (1995) present a general model of inductive bias, as well as an analysis of systems that adapt their biases.

Whenever SN locates a good solution, it depresses the fitness landscape at all points within some radius of that solution (in our implementation, we penalize the region corresponding to the pattern uniformly). Packard (1989) proposed a similar penalty function.

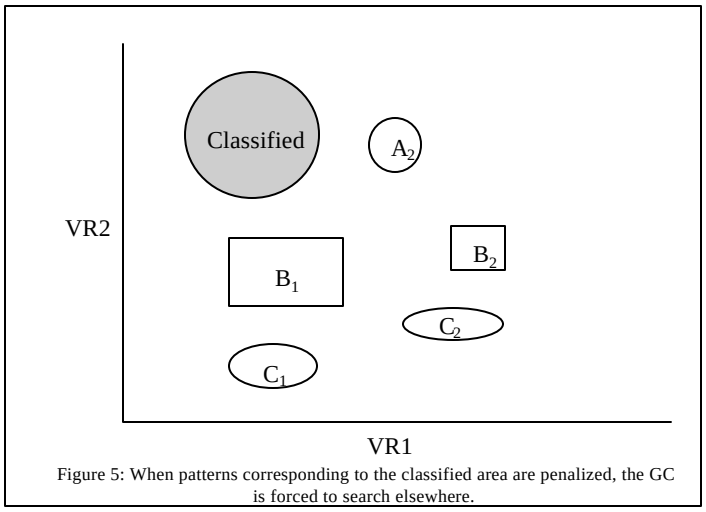

Figure 5: When patterns corresponding to the classified area are penalized, the GC is forced to search elsewhere.

4.4.2 Removing Classified Areas by Data Reduction: DR

The most commonly used heuristic for inductive strengthening is the *covering* heuristic. Made popular by the family of separate-and-conquer rule learning algorithms (Furnkranz, 1999), once a good rule is found the covering heuristic removes the examples it covers from the data set. Since these algorithms determine rule interestingness (and therefore search direction) by statistics on rule coverage, removing the covered examples implicitly leads the search elsewhere.

As an alternative to sequential niching, we can apply the covering heuristic to genetic search. Specifically, after the search converges on a pattern, instead of penalizing an *area* as classified when calculating the fitness function, the *data* corresponding to the area are removed from further consideration. This is similar to an approach used by Sikora and Shaw (1994) for inductive strengthening in their genetic learning algorithm. Let us compare data reduction to sequential in the context of our genetic search.

With sequential niching, as the larger areas become classified, subsequent search tends to produce more specialized patterns. Figure 6 shows why. Suppose the dark-shaded area is not yet classified. The algorithm will tend to produce low-support patterns (small circle), because larger areas (higher support) have a higher chance of hitting penalty areas. This is an unintended consequence—we do not want to restrict our search from the new, large pattern, we just want to focus it away from the already-found large pattern.

In contrast, discarding data associated with a discovered pattern has a different effect. Since there is no penalty area, the new, large pattern will not be penalized. Since it has larger support, if the confidences are comparable, it will be preferred over the smaller pattern. Of course, the confidences may not be the same, and indeed, the pattern could produce higher misclassification rates on the *removed* data. The effect of this is that the algorithm continues to produce more general patterns (corresponding to the larger area in Figure 6) later in the search, that is, patterns that tend to have higher support, but confidence may suffer when applied to the original data. Also, there is less control on overlap with other rules than in sequential niching.

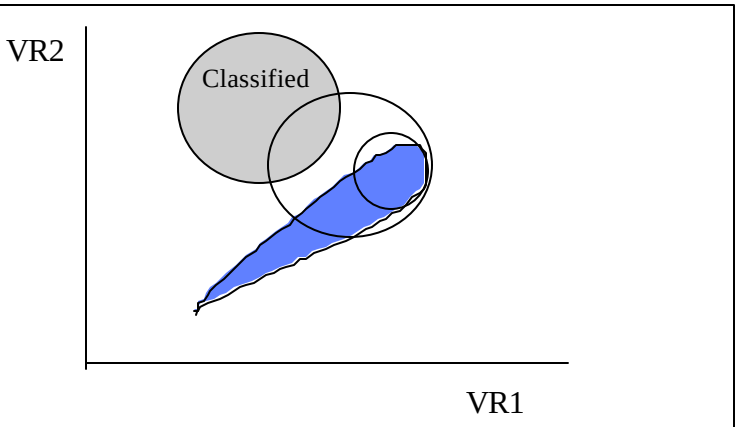

Figure 6: When the larger area has more chance of hitting an already classified area and getting penalized, the GC will tend to favor more specific, i.e. patterns with less support.

Thus, one may understand the differences between penalizing the area versus removing the data as: the former is more likely to produce non-overlapping rules whereas the latter produces a hierarchy of rules, as shown in Figure 7, which potentially may overlap.

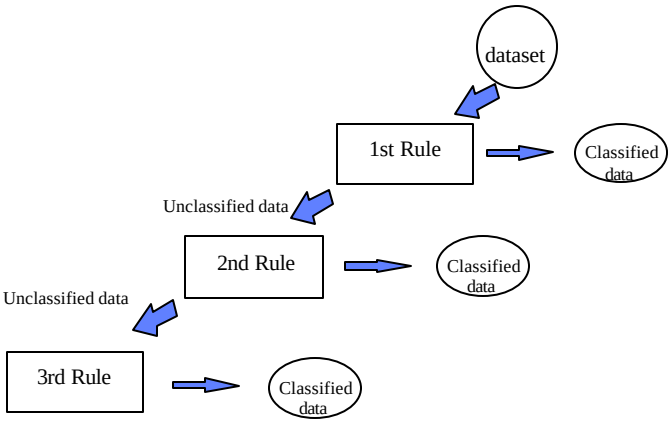

Figure 7: The patterns from are not independent but form a hierarchy of rules

One obvious drawback to removing data is that the algorithm may discard both correctly classified data *and* misclassified data as shown in Figure 8. In contrast, since sequential

niching does not discard data, there is a chance that data misclassified by an earlier pattern (A) will be correctly classified by another pattern (B) as shown in Figure 9. Another way of looking at this is that discarding data removes support for schemata that overlap with the one for which data were removed.

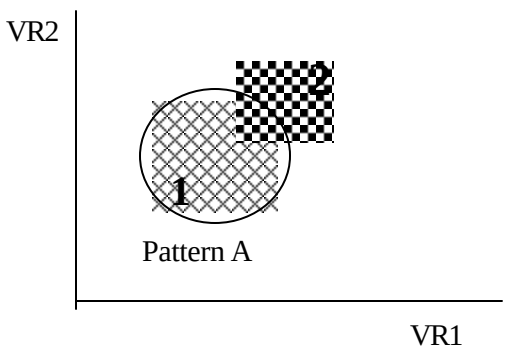

Figure 8: Data Reduction removes the area covered by pattern A. Thus, part of the misclassified data of '2' is lost

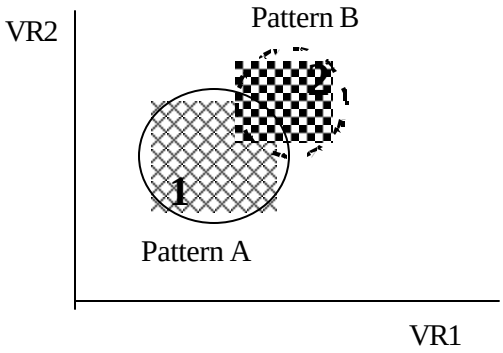

Figure 9: Sequential niching keeps data misclassified by pattern A, data '2'. This area can later be correctly classified by pattern B

5. Genetic Rule Discovery with Entropy Reduction and Inductive Strengthening

A seemingly obvious solution to the problem of data reduction is to be more judicious in inductive strengthening, for example, only discard examples that are *correctly* classified by the already-learned rules (Provost & Buchanan, 1992). However, the smaller sample size that results from the data reduction also tends to increase variance of the dependent variable, thereby lowering the fitness of the pattern. This undesirable phenomenon is not relegated to separate-and-conquer rule learners, but also applies to tree induction algorithms, which also split up the data based on "rules" learned so far (via the higher-level splits). What we would like is for an inductive strengthening heuristic to be used to guide the search, but for the statistics used to determine fitness to be gathered over the entire database. This is the philosophy taken by rule learners such as RL (Clearwater & Provost, 1990) and CWS (Conquer Without Separating) (Domingos, 1996a).

Our genetic learning approach incorporates these ideas as well as those that allow TI algorithms to find good splits quickly. Our Genetic Learner Overlaid With Entropy Reduction (GLOWER) recursively feeds entropy-reducing splits into the genetic search. The splits used are those that best discriminate the class membership of the dependent variable. In this way, the genetic algorithm incorporates promising "pieces of patterns," the building blocks, into the search and "spreads" them to other patterns through crossover and mutation.

One of the interesting features of this strategy is that, ironically, the genetic algorithm is able to make much better use of entropy reduction splits than can greedy search algorithms that split the data. With recursive partitioning, splits lower down in the tree

apply to smaller subsets of the data, thereby producing a higher variance on the dependent variable class distribution because of smaller sample size. In contrast, the genetic algorithm gives each split a chance to be evaluated on the entire database.[7] Thereby, parts of trees can be combined with others, leading to patterns that are much harder to find with irrevocable splits. In sum, the genetic algorithm is able to use the efficient sifting of the data that tree induction methods generate in order to focus its search to promising areas of the search space, but then allows the genetic search to assemble the building blocks into even better models.

This hybrid search method uncovers promising components of patterns by comparing the distributions of class membership based on splits on an independent variable in the same way that TI classifiers work. However, instead of partitioning the data set irrevocably, it simply introduces the split as a building block into the search, enabling it to *interact* with other promising building blocks. When combined with niching, this allows GLOWER to focus its search dynamically based on the state of the search.

5.1. Comparison of GLOWER variants: experimental design

We now report results from a set of experiments where our objective was to evaluate several variations of GLOWER based on the heuristics described above, and to pick a good one for comparison with TI and rule learning algorithms (section 6). For these preliminary experiments, three data sets were used. The first two, waveform and character recognition, are from the UCI repository [UCI, 1995]. They represent noisy problems with numeric attributes and overlapping classes. The third data set is from the financial arena: the prediction of weekly returns of stocks in the S&P500.

5.1.1. Data

The first data set consists of records corresponding to the three types of waveforms as shown in Figure 10. It was also used by Breiman *et al.* (1984) in evaluating CART.

[7] This is not always true, but we ask the critical reader to bear with us for the moment.

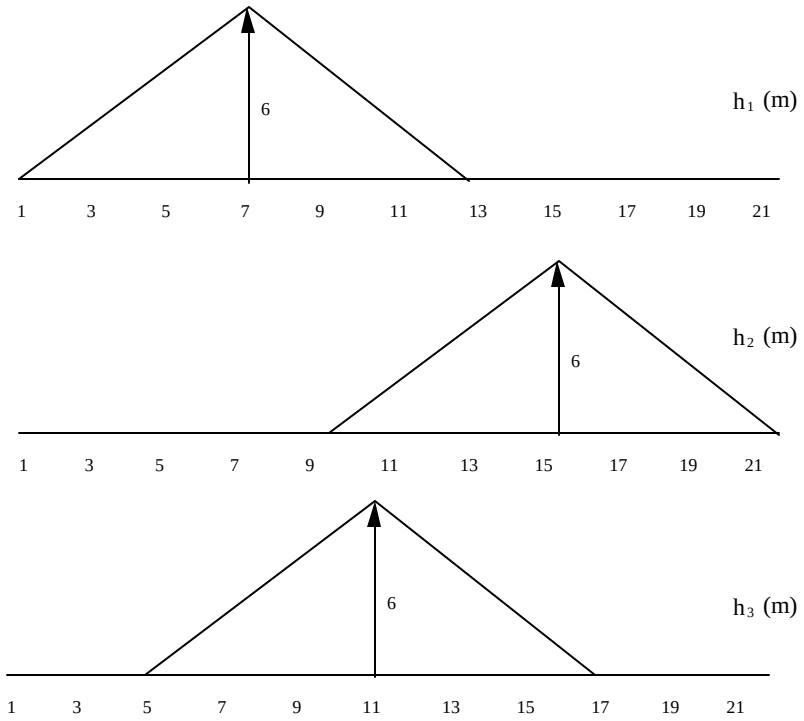

Figure 10: Three Waveforms, each with a 21 dimensional measurement

Each class consists of a random convex combination of two of these waveforms, sampled at the integers with noise added. There are 21 real-valued independent variables.

Type 1 vector X is defined as:

$$X_m = u_x * h_1(m) + (1 - u_x) * h_2(m) + \varepsilon_{xm}, \quad m = 1,2, ..., 21$$

Type 2 vector Y is defined as:

$$Y_m = u_y * h_2(m) + (1 - u_y) * h_3(m) + \varepsilon_{ym}, \quad m = 1,2, ..., 21$$

Type 3 vector Z is defined as:

$$Z_m = u_z * h_3(m) + (1 - u_z) * h_1(m) + \varepsilon_{zm}, \quad m = 1,2, ..., 21$$

where u_x, u_y, and u_z are uniformly distributed in [0,1] and ε_{xm}, ε_{ym}, and ε_{zm} are normally distributed, N(0,1). This data set contains 5000 records.

The second data set consists of records where the dependent variable is a letter, from A to Z. There are 16 independent variables denoting attributes such as shapes and their positions and sizes. This data set contains 20000 records, with unequal numbers of examples (cases) in each class. This is a challenging problem, in part due to the large number of classes (26), and a significant amount of overlap in the patterns of independent variables corresponding to the various classes.

The third data set consisted of prices of financial instruments constructed from the S&P 500 universe of stocks between January 1994 and January 1996, excluding mergers and acquisitions. The objective was to predict returns one week ahead. Two classes were created for the dependent variable, namely, up or down, depending on whether the future

return was above a specified "high" threshold, or below a specified "low" threshold (we ignored the "neutral" cases for this experiment). The classes had roughly equal prior distributions. Approximately 40 standard technical analysis indicators such as moving averages and volatilities were computed as in Barr and Mani (1994), which served as the independent variables. There is a degree of subjectivity in the choice of the prediction period as well as the various possible data transformations chosen, and other, possibly better, transformations could have been chosen. However, all of the GLOWER variants had to deal with the shortcomings of the problem formulation and the data, and in this sense, they were on an equal footing.

5.1.2 Sampling Design

Each of the initial data sets was partitioned into 3 subsets: in-sample training data, in-sample testing data (for the evaluations internal to GLOWER), and out-of-sample testing data. Table 3 shows the size of each subset for the three data sets.

	In-sample training data	**In-sample testing data**	**Out-of-sample testing data**	**total**
Data set 1: Waveform Recognition	3,010	770	1,220	5,000
Data set 2: Character Recognition	12,000	4,000	4,000	20,000
Data set 3: Estimating returns of S&P500 stocks	11,600	3,200	3,200	18,000

Table 3: Data Samples

We chose this multi-level separation because of the several dangers of overfitting. Most obviously, the in-sample/out-of-sample division is important so that the model can be evaluated in terms of consistency. Within the in-sample data, the training and testing division is important, similarly, to allow the genetic algorithm to evaluate the consistency of a hypothesis being tested.

5.2. Assessment of GLOWER Variants

In this section, we present the results from applying to these data a more-or-less standard genetic rule discovery algorithm and three enhancements to it, based on the inductive strengthening techniques described above. In particular, we report results from "standard" parallel niching, which we consider as the baseline, and the following:

- sequential niching
- sequential niching plus entropy reduction
- data reduction plus entropy reduction[8]

Figures 11a through 11c show the results from the first set of experiments. The waveform recognition problem was easy for all algorithms, producing high support and confidence.

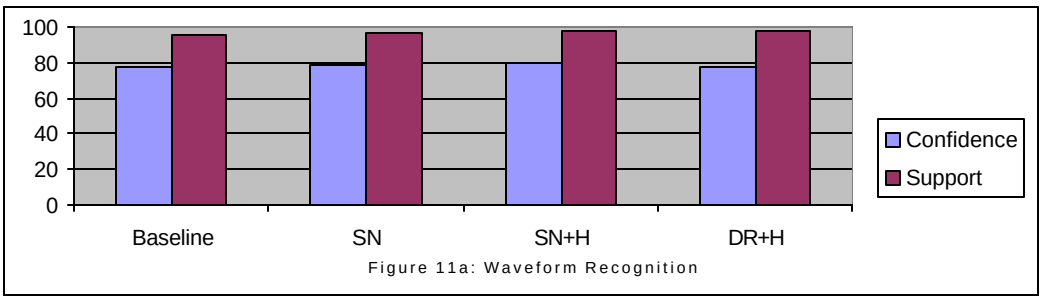

Figure 11a: Waveform Recognition

Apparently, for easy problems, the choice of inductive strengthening technique makes little difference.

The character recognition problem is a little harder. The dependent variable comprises 26 classes, 'A' through 'Z'. All variations of the genetic algorithm did well on confidence, but there is a marked increase in support when the entropy reduction heuristic is used in conjunction with sequential niching or data reduction. Figure 11b shows that the algorithm SN+H, sequential niching with the entropy reduction heuristic, performed the best in terms of confidence (92%) and had a high degree of support (83%).

As would be expected, the S&P stock prediction problem was the hardest, because there is very little structure in the data as is apparent in Figure 11c. The results from this experiment point to the difficulties in market forecasting and illustrate the tradeoffs among the techniques for finding patterns in this domain. We should note that by viewing the results of the partial models learned by GLOWER we see a very different picture from that presented by techniques that try to model the problem as a whole (not shown). In the latter case, the accuracy of just about any model hovers around the accuracy of a knowledge-free model (depending on the problem formulation, one may choose random guessing, buy-and-hold, or some other simple strategy). Any structure contained in the model is swamped by random variation. Moreover, realistic evaluations (from the perspective of actually making investments) involve costs of trading. These

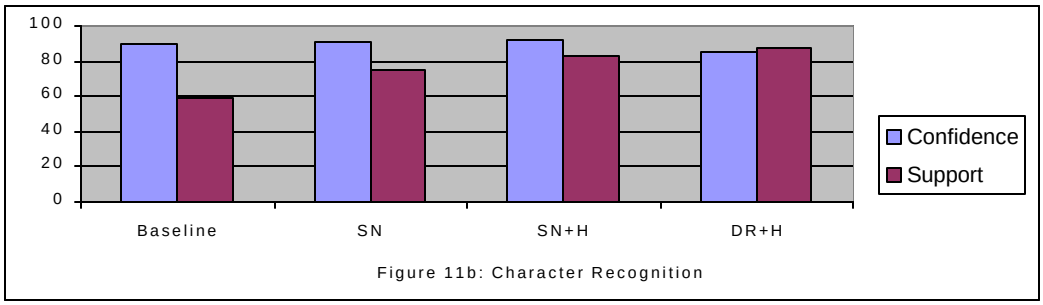

Figure 11b: Character Recognition

factors determine the required balance between confidence and support. Both the confidence and the support must be high enough to provide a good tradeoff of risk and reward.

As is evident from Figure 11c, all variations of the genetic learning algorithm had very low support for this problem. The genetic algorithms, in effect, indicated that they would not make a prediction most of the time. This makes sense since it is very difficult to make an accurate directional stock-market prediction. The genetic algorithm is unable to find rules that perform well consistently on its in-sample testing data. However, the

overlaid heuristics did improve the support significantly, with entropy reduction resulting in the highest support when combined with either sequential niching or data reduction. While the baseline algorithm produced a support of only 0.7%, sequential niching improved this to 1.7%, with a confidence of 72%. As with the other data sets, SN+H and DR+H provided the best overall results, with the former resulting in rules with a

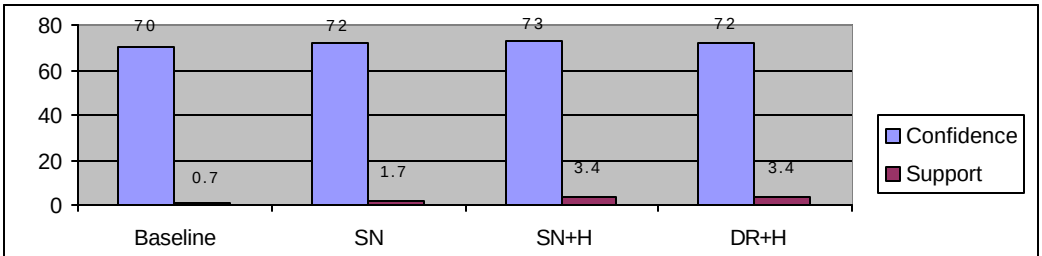

Figure 11c: Estimation of monthly Stock Returns

confidence of 73% and support of 3.4%. In effect, the overlaid heuristics provided a *fivefold* increase in coverage as compared to the baseline genetic algorithm, while increasing confidence marginally as well.

In sum, the experiments show that SN+H and DR+H generally are preferable to the baseline algorithm and to SN alone. The results do not support a strong distinction between SN+H and DR+H, suggesting that niching and separate-and-conquer tend to be similar in terms of their inductive strengthening ability. However, since SN+H had marginally better performance in terms of confidence on the latter two data sets, we chose it for the next study that compares GLOWER to tree and rule induction algorithms.

6. Comparison of GLOWER to Tree and Rule Induction

For the main set of experiments we used the best performing GLOWER variant from the previous study (SN+H) and applied it to another financial problem, namely, to predict "earnings surprises." This problem has been studied extensively in the business literature. The objective is to classify a company's next earnings report as "positive," "negative," or "neutral," depending on whether one predicts that these earnings will significantly exceed, fall short, or be in line with the average reported expectations of a set of analysts. It has been shown that positive surprise companies tend to produce returns after the announcement that are in excess of the market, whereas the opposite is true for the negative surprise firms. Several explanations have been proposed for this phenomenon. The details of the earnings surprise research can be found in Chou (1999).

The data set for this study consisted of a history of earnings forecasts and actual reported earnings for S&P500 stocks between January 1990 and September 1998. These were used to compute *earnings surprise*, which is "positive" if actual announced earnings exceed the analyst consensus estimates by a specified threshold (half a standard deviation of estimate), "negative" if they fall short, and "neutral" otherwise. The prior class distribution was roughly 13.5 percent positive, 74 percent neutral, and 12.5 percent

negative surprises. In other words, companies report earnings that are mostly within half a standard deviation of expectations.

Approximately 60 independent variables were chosen based on commonly used indicators in the fundamental and technical valuation analysis literature (Graham and Dodd, 1936; Achelis, 1995; Madden, 1996). The technical variables chosen correspond to price trend and volatility, whereas the fundamental variables are based on financial statements, such as historical cash flows and earnings. The specific indicators used are described in Chou (1999). The objective in this problem is to predict 20 days before the actual earnings announcement whether a specific company will report a positive, neutral, or negative earnings surprise as defined above. Predicting earnings surprise is one way of forecasting future returns. The data set sizes used are listed in Table 4, similarly to Table 3 (described above).

	In sample training data	In sample testing data	Out of sample testing data	Total
Predicting earnings surprise	14,490	9,650	12,164	36,204

Table 4: Data Samples

We compare GLOWER using sequential niching with the entropy reduction heuristic (SN+H) to a standard tree induction algorithm, CART (Breiman et.al, 1984), and a rule learning algorithm, RL (Clearwater and Provost, 1990).

One way of defining misclassification in earnings surprise prediction is if an actual positive surprise was predicted as neutral or negative, or if an actual negative surprise was predicted as neutral or positive. Another way, which we believe to be better for this domain, is to define a misclassification only if a positive is classified as negative and vice versa. Economic arguments support this view, since the detriment associated with misclassifying a positive as a negative is significantly greater than that associated with misclassifying a positive as a neutral (and similarly for negatives). Our results (see Figures 12 and 13) use confidence with this latter interpretation of misclassification, although the comparative results are not materially different with the alternative interpretation.

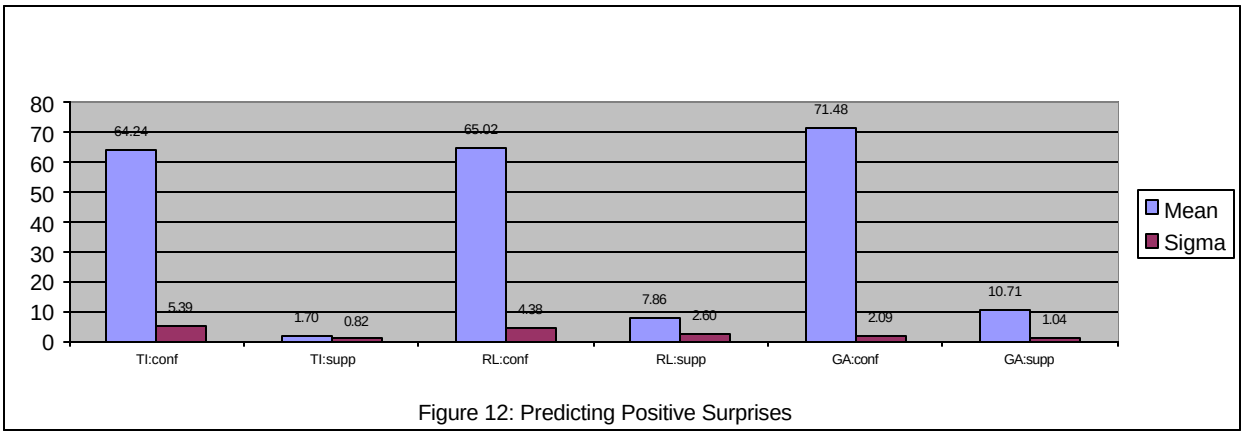

Figure 12: Predicting Positive Surprises

Figure 12 shows the results from tree induction, rule learning and genetic learning for predicting positive surprises based on fifty sets of runs. For each of the three methods, the figures show the means and standard deviations for confidence and support for each of the methods. When applied pairwise, all means are different at the 0.0001 level of significance. Confidence and support increase from left to right and the variances of these measures decrease.

Figure 13 shows the corresponding results for negative surprises. Again, all means considered pairwise are different at the 0.0001 level of significance except for GLOWER versus TI confidence means which are different at the 0.15 level of significance.

A number of things stand out in the results. Most importantly, perhaps, the results are in line with what we hypothesized in Figure 1: the various methods perform along the spectrum of thoroughness of search that was shown in the figure. GLOWER is indeed more thorough in its search than rule induction which is in turn more thorough than tree induction.

More specifically, we can see from Figure 12 that GLOWER outperforms the other two in terms of both confidence and support for predicting positive surprises. For support, GLOWER has *six* times the support of the TI algorithm and one and a half times that of rule learning. Also its variances on confidence and support are lower on average. For negative surprises (Figure 13), the confidence levels are similar, but GLOWER is again significantly better in terms of support (more than double that of tree induction and roughly double that of rule induction). In effect, GLOWER's rules cover anywhere between two and six times more of the problem space than its competitors' without sacrificing accuracy. Clearly, the rules from its competitors are more specialized than is necessary. For problems with weak structure, this is a significant difference. The result is in line with our prior hypothesis described in Figure 1.

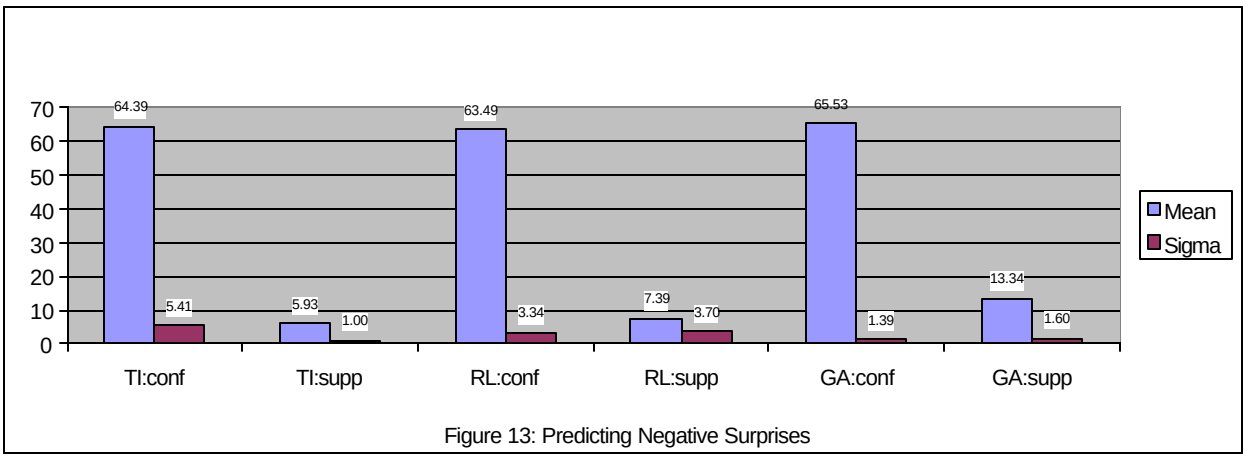

Figure 13: Predicting Negative Surprises

An unexpected result was that the genetic learning algorithm appears to be much more suited to capturing symmetry in the problem space as is evident in its relatively uniform performance at the *opposing* ends of the problem space. Specifically, the coverage for the

TI algorithms positive surprises is 5.93% compared to 1.7% for negative surprises, while the comparable numbers for the genetic learning algorithm are 13.3 and 10.7 percent respectively. TI and RL find it harder to predict a negative earnings surprise than a positive one. The result should make us question whether predicting a negative surprise is harder, either inherently, or because the available data do not represent this phenomenon adequately, making it harder for an algorithm to discover. One interpretation of the results, based on observing the rules, is that the greedy heuristic picked up on one major effect, momentum, which is correlated with surprise, but only weakly. Further, this variable was not negative often enough in the data, making it hard to find negative surprise rules. However, GLOWER was able to identify other variables, such as cash-flow-based measures from balance sheets, in order to identify additional situations that were indicative of impending negative surprises. If this is the case, it would be consistent with our analysis in section 5 about the ability of GLOWER to splice together different building blocks into potentially useful patterns and its ability, via sequential niching and recursive entropy reduction splits, to discover the underlying patterns in the data. Verifying whether this indeed is the case is future work for us, and we revisit this issue in the next section.

7. Discussion

Why are the results reported above interesting? Is there something about the genetic learning algorithm that enables it to perform better for financial prediction problems?

We have already discussed that relationships among variables in the financial arena tend to be weak. A correlation coefficient higher than 0.1 between an independent and dependent variable is uncommon. Also the correlation is typically nonlinear: it may be strong in the tails and non-existent everywhere else. As we mentioned earlier, a positive analyst revision on a stock may have no impact on it *unless* the revision exceeds some threshold. Even more importantly, the impact may be magnified or damped by other variables, for example, whether the stock belongs to a high- or low-growth sector. In such situations, we might achieve better insight into the domain, and achieve higher overall predictability, by considering interaction effects. The learning algorithm should be able to find such effects and still produce patterns with reasonable support.

GLOWER uses entropy reduction and inductive strengthening heuristics to find the "interesting seeds" in the problem space and then expands them as much as possible. In doing so, it also combines these seeds in many ways, thereby capturing interaction effects among variables. Methods such as tree and rule induction are good at finding interesting seeds, but often are not able to exploit them any further. For tree induction algorithms this is largely because the process of splitting the data results in misclassifications from which they cannot recover, and in higher variances for the dependent variable, which makes it difficult to find interesting interaction effects. Rule learning does better, but tends not to explore combinations of variable splits that may be individually suboptimal, but be collectively better than the combination of the best univariate splits. In contrast, the genetic algorithm is able to mix and match seeds and it implicitly performs local

sensitivity analyses. If a combination turns to be potentially robust, i.e. produces consistently good results across training and testing sets, this becomes another seed that can be explored and refined further.

Because the traditional methods are reasonable at finding some of the initial seeds, we found it worthwhile using them as part of the genetic search. In this way, we are able to combine the speed of these methods with the more comprehensive search of the genetic algorithm. In effect, the genetic learning algorithm is guided by the splits fed to it by entropy reduction. Indeed, the range-oriented representation for numeric variables used by CART works well for GLOWER since they use identical representations for their concept class. While non-backtracking-based methods such as CART tend not to recover from the misclassifications that are inevitable with such methods, the genetic algorithm is less restricted since every split is evaluated against the entire database in conjunction with many other interesting splits.

From a practical standpoint, for financial problems where there is weak structure and the objective is to develop investment strategies, it is important that both accuracy and support be high enough for the corresponding rules to be "actionable." Consider the earnings surprise problem. If our objective is to develop a "long/short" strategy, it is important that there be *enough* stocks to "long" and enough to "short." The CART output in the example, where support is 6% and 1.7% respectively, when applied to the S&P500 h universe would result in 30 longs and about 8 shorts. These numbers are on the low side and are highly asymmetric, and an investment professional looking for a "market neutral" strategy (where equal capital is committed to the long and short sides) would be hesitant to apply such a strategy. The genetic algorithm, in contrast, would produce about 65 longs and 55 shorts, probably much more desirable strategy to implement.

In summary, the genetic learning algorithms we have described employ useful heuristics for achieving higher levels of support while maintaining high accuracy. They also enable us to incorporate domain-specific evaluation criteria, such as how to trade off confidence and support. In practice, using such patterns, whether for trading, marketing, or customer profiling, requires taking into account transaction costs and other problem specific factors such as how much one wants to trade, how many mailings to send out, and so on. This involves specifying the appropriate tradeoffs between accuracy and coverage for a particular problem. The genetic learning algorithm provides us with the knobs for choosing the approximate accuracy and coverage levels we are interested in, and tuning the search accordingly.

So, should we always prefer to use the genetic algorithm over other algorithms such as tree induction? Not really. The genetic algorithm is about two to three orders of magnitude slower than a TI algorithm, which is not surprising considering the number of computations it has to perform. Roughly speaking, an analysis that takes CART two minutes takes the genetic algorithm many hours. The practical implication of this is that the genetic algorithm is not a rapid experimentation tool. It may make more sense to use greedy search algorithms during the early stages of analysis in order to get a quick

understanding of the more obvious relationships if possible, and to use this information to focus on a limited set of variables for the genetic search.

On a technical note, we view the results as contributing a useful data point to the research on comparing alternative approaches to pattern discovery. In this case, our choice of techniques has been driven by the need for high *explainability* to end users, while achieving high accuracy. In other words, our objective is not simply prediction, but to build an *explicit* model of the domain that can be critiqued by experts. This consideration has been a major factor in using the rule-based concept-class representation of the genetic algorithm. It has also discouraged us from using methods that produce outputs that are harder to interpret. Decision makers in the financial arena tend not to trust models they cannot understand thoroughly unless such models guarantee optimality. Rules are easy to understand and evaluate against expert intuition. This makes rule induction algorithms, such as GLOWER, attractive pattern discovery methods.

More generally, explainability is an important consideration in using knowledge discovery methods for theory building. This has been the central focus of the current research. However, theory building requires taking into consideration other features of model outputs such as their relative *simplicity.* Specifically, we have not remarked about the number of conditions in the rules for which we have presented the results. The more conditions there are, the more potential for overfitting, and the less transparent the theory. In our experiments we typically restrict GLOWER to finding patterns with at most four or five variables in order to make the outputs easier to interpret. The tree induction algorithm tended to produce more specialized (syntactically) rules than GLOWER. The tree and rule induction algorithms both produced rules that individually had lower support than those produced by GLOWER, for comparable confidence levels. We have not reported on this aspect of theory building because the results in this area are somewhat preliminary. Similarly, we have not taken into account other considerations, such as extent to which the outputs produced were consistent with prior background knowledge and so forth. In future research we will report on these aspects of theory building with different knowledge discovery methods.

8. Conclusion

This paper has a number of research contributions. We claim that for hard problems, genetic learning algorithms, appropriately “fixed” up, are more thorough in their search than other rule learning algorithms. Our results support this claim. GLOWER indeed is less restricted by greedy search biases, and for problems with weak structure or variable interactions, it is precisely the subtle relationships that are useful discoveries.

Genetic algorithms have a number of inherent benefits that we have exploited in GLOWER, including the flexibility to accommodate variable fitness functions. We have borrowed heuristics from tree induction and rule induction to fix genetic search's inherent weaknesses, viz., randomness and myopia in search. In particular, GLOWER takes advantage of the strengths of entropy reduction and inductive strengthening methods to focus the search on promising areas and to refocus after interesting individual patterns have been found.

More generally, the research provides a view of the thoroughness of rule space search, in particular for financial data. It also provides comparison and contrast of the heuristics used by various different rule-mining techniques. Such a comparison is important in our ongoing quest to understand the true strengths and weaknesses of the various rule-mining methods on practical problems.

References

Achelis, S.B., *Technical Analysis From A to Z.,* Chicago: Irwin, 1995.

Atiya, A., An Analysis of Stops and Profit Objectives in Trading Systems, *Proceedings of the Third International Conference of Neural Networks in Capital Markets (NNCM-95),* London, October 1995.

Barr, D., and Mani, G., Using Neural Nets to Manage Investments, *AI Expert*, February 1994.

Bauer, R. J., *Genetic Algorithms and Investment Strategies*, John Wiley & Sons, 1994.

Beasley, D., Bull, D.R., and Martin, R.R. A sequential niche technique for multimodal function optimization. Evolutionary computation, 1(2), 101-125.

Blake, C., Keogh, E., and Merz, C.J.,, Repository of machine learning databases, University of California, Irvine, Dept. of Information and Computer Sciences, 1998.

Breiman, L., Friedman, J., Olshen, R., and Stone, C., *Classification and Regression Trees*, Wadsworth, Monterey, CA, 1984.

Cartwright, H. M. and Mott, G. F. Looking around: using clues from the data space to guide genetic algorithm searches. Proceedings of the fourth international conference on genetic algorithms, 1991.

Chou, Dashin, The Relationship Between Earnings Events and Returns: A Comparison of Four Nonlinear Prediction Models, Ph.D Thesis, Department of Information Systems, Stern School of Business, New York University, 1999.

Clark, P., and Niblett, T., The CN2 Induction Algorithm, *Machine Learning*, 3, pp 261-283, 1989.

Clearwater, S., and Provost, F., RL4: A Tool for Knowledge-Based Induction, Proceedings of the Second International IEEE Conference on Tools for Artificial Intelligence, pp. 24-30, 1990.

Cohen, William W., and Singer, Yoram., A Simple, Fast, and Effective Rule Learner, Proceedings of the Sixteenth National Conference on Artificial Intelligence,.American Association for Artificial Intelligence 1999 (AAAI-99) pp. 335-342.

Deb, K. and Goldberg, D. E. An investigation of Niche and Species Formation in Genetic Function Optimization. Proceedings of the third international conference on genetic algorithms, 1989.

DeJong, Ken., Evolutionary Computation for Discovery, *Communications of the ACM*, volume 42, number 11, pp. 51-53, 1999.

Dhar, V., and Stein, R., *Seven Methods for Transforming Corporate Data Into Business Intelligence*, Prentice-Hall, 1997.

Domingos, P., Unifying Instance-Based and Rule-Based Induction, *Machine Learning*, 24, 141-168, 1996a.

Domingos, P., Linear Time Rule Induction, Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, AAAI Press, pp. 96-101, 1996b.

Forgy, L., RETE: A Fast Algorithm for Many Pattern / Many Object Pattern Matching, *Artificial Intelligence*, 19, pp. 17-37, 1982.

Friedman, J. H. Local learning based on recursive covering. Dept. of Statistics, Stanford University, 1996.

Furnkranz, J., Separate-and-Conquer Rule Learning, *Artificial Intelligence Review*, volume 13, number 1, pp. 3-54, 1999.

George, E. I., Chipman, H. and McCulloch, R. E. Bayesian CART. Proceedings: Computer Science and Statistics 28th Symposium on the Interface. Sydney, Australia, 1996.

Goldberg, D.E. *Genetic algorithms in search, optimization, and machine learning*. reading, MA: Addison-Wesley, 1989.

Goldberg, D. E., Deb, K. and Horn, J., Massive multimodality, deception and genetic algorithms, *Parallel Problem Solving from Nature*, 2, Manner, R. and Manderick B., eds., Elsevier Science, 1992.

Goldberg, D. E. and Richardson, J. Genetic algorithms with sharing for multimodal function optimization, *Proceedings of the Second International Conference on Genetic Algorithms*, 1987.

Graham, B., and Dodd, D., *Security Analysis*, McGraw-Hill, 1936.

Grefenstette, J.J. Incorporating problem specific knowledge into genetic algorithms. In Davis, L., editor, Genetic Algorithms and Simulated Annealing, pages 42-60, Los Altos, CA. Morgan Kaufmann, 1987

Hekanaho, J. Background knowledge in GA-based concept learning. Proceedings of the thirteen international conference on machine learning, 1996.

Hong, J., Incremental Discovery of Rules and Structure by Hierarchical and Parallel Clustering, in *Knowledge Discovery in Databases*, Piatetsky-Shapiro and Frawley, eds, AAAI Press,Menlo Park CA 1991.

Holland, J. H. *Adaptation in natural and artificial systems.* Ann Arbor: The University of Michigan Press. 1975.

Holland, J. H. *Adaptation in natural and artificial systems.* An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. MIT press, 1992.

Janikow, C.Z. A knowledge-intensive genetic algorithm for supervised learning. machine Learning, 13:189-228.

Jensen, David., and Cohen, Paul R., Multiple Comparisons in Induction Algorithms, *Machine Learning*, to appear in 2000.

Lim, Tjen-Jen., Loh, Wei-Yin., and Shih, Yu-Shan Shih., A Comparison of Prediction Accuracy, Complexity, and Training Time of Thirty-three Old and New Classification Algorithms, *Machine Learning*, to appear in 2000.

Madden B., The CFROI Life Cycle, *Journal of Investing*, volume 5, number 1, Summer 1996.

Mahfoud, S. W. A comparison of parallel and sequential niching methods. *Proceedings of the sixth international conference on genetic algorithms*, 1995.

Mahfoud, S. W. Niching methods for genetic algorithms. Urbana: U. of Illinois, Illinois Genetic Algorithms Lab. 1995.

Michalski, R., Mozetec, I., Hong, J., and Lavrac, N., The Multi-Purpose Incremental Learning System AQ15 and Its Testing to Three Medical Domains, in Proceedings of the Sixth National Conference on Artificial Intelligence, 1041-1045, Menlo Park, CA, 1986.

Michie, D., Spiegelhalter, D. J. and Taylor, C. C. *Machine Learning, Neural and Statistical Classification*, Ellis Horwood Ltd., 1994.

Mitchell, T.M., The need for biases in learning generalizations, Report CBM-TR-117, Computer Science Department, Rutgers University, 1980.

Murthy, Sreerama K., Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey, *Data Mining and Knowledge Discovery*, 2 (4):345-389, December 1998.

Oei, C. K., Goldberg, D. E. and Chang, S., Tournament Selection, Niching and the Preservation of Diversity, Urbana: U. of Illinois, Illinois Genetic Algorithms Lab. 1991.

Packard, N., A Genetic Learning Algorithm, Tech Report, University of Illinois at Urbana Champaign, 1989.

Provost, F.J. and Buchanan, B.G.}, Inductive Policy: The pragmatics of bias selection, *Machine Learning*, volume 20, pp. 35-61, 1995.

Provost, F. and Buchanan, B., Inductive Strengthening: The effects of a simple heuristic for restricting hypothesis space search, In K.P. Jantke (ed.), *Analogical and Inductive Inference* (Lecture Notes in Artificial Intelligence 642). Springer-Verlag, 1992.

Provost, F., Aronis, J., and Buchanan,B., Rule-space search for Knowledge-based Discovery, Report #IS 99-012, IS Dept., Stern School, NYU.

Quinlan, J., *Machine Learning and ID3*, Morgan Kauffman, Los Altos, 1996.

Sikora, R. and Shaw, M. J., A double-layered learning approach to acquireing rules for classification: Integrating genetic algorithms with similarity-based learning. *ORSA Journal on Computing* 6(2), pp. 334-338, 1994.

Smythe, P., and Goodman, R., Rule Induction Using Information Theory, in *Knowledge Discovery in Databases*, Piatetsky-Shapiro and Frawley, eds, AAAI Press,Menlo Park CA 1991.

UCI Repository of machine learning databases [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science, 1995.