Applied Artificial Intelligence

ISSN: 0883-9514 (Print) 1087-6545 (Online) Journal homepage: www.tandfonline.com/journals/uaai20

Financial forecasting using genetic algorithms

Sam Mahfoud & Ganesh Mani

To cite this article: Sam Mahfoud & Ganesh Mani (1996) Financial forecasting using genetic algorithms, Applied Artificial Intelligence, 10:6, 543-566, DOI: 10.1080/088395196118425

To link to this article: https://doi.org/10.1080/088395196118425

FINANCIAL FORECASTING USING GENETIC ALGORITHMS

SAM MAHFOUD and GANESH MANI
LBS Capital Management, Inc., Clearwater, Florida, USA

A new genetic-algorithm-based system is presented and applied to the task of predicting the future performances of individual stocks. The system, in its most general form, can be applied to any inductive machine-learning problem: given a database of examples, the system will return a general description applicable to examples both within and outside the database. This differs from traditional genetic algorithms, which perform optimization. The genetic algorithm system is compared to an established neural network system in the domain of financial forecasting, using the results from over 1600 stocks and roughly 5000 experiments. Synergy between the two systems is also examined.

This study presents a new system that utilizes genetic algorithms (GAs) to predict the future performances of individual stocks. More generally, the system extends GAs from their traditional domain of optimization to inductive machine learning or classification. The overall learning system incorporates a GA, a niching method (for finding multiple solutions), and several other components (discussed in the section entitled Genetic Algorithms for Inductive Learning).

Time-series forecasting is a special type of classification on which this study concentrates. Specifically, for any financial time series related to the performance of an individual stock, the goal is to forecast the value of the time series k steps into the future. The experiments of this study forecast the relative return of a stock 12 weeks into the future. We define a stock's relative return as the stock's return minus the average return of the over 1600 stocks we model. We make predictions for all 1600+ stocks at three different points in time and summarize the results.

As a benchmark, the GA system is compared to an established neural network (NN) system (Mani & Barr, 1994) using the same 1600+ stocks and three points in time. (We have used the NN system and its predecessors to forecast stock prices and manage portfolios for approximately 3 years.) We examine the potential synergy from combining the GA and NN forecasts, as well as other ways in which the two algorithms complement each other.

The remainder of this article discusses inductive machine learning, casting financial forecasting as an inductive machine-learning problem; reviews genetic

The authors thank Dean Barr, K. K. Quah, and Doug Case for their advice and assistance, and Steve Ward of Ward Systems Group for help with neural network implementations. The authors also thank the referees for their suggestions.

Address correspondence to Sam Mahfoud, LBS Capital Management, Inc., 311 Park Place Boulevard, Suite 330, Clearwater, FL 34619, USA. E-mail:sam@lbs.com

Applied Artificial Intelligence, 10:543–565, 1996

0883-9514/96 $12.00 + .00

algorithms; examines genetic algorithms in inductive machine learning and financial forecasting; explains the GA-based system of this study; discusses the chosen applications domain—predicting the performances of individual stocks; presents two sets of experiments and their associated results; examines the results as well as experimental biases; and presents paths for future research.

FINANCIAL FORECASTING AS INDUCTIVE MACHINE LEARNING

This section briefly reviews inductive machine learning and proceeds to cast the problem of financial time-series forecasting as a specific type of inductive machine learning. Inductive learning (Michalski, 1983) can be defined as acquiring concepts through examining data items. It is the similarities among various data items that allow an inductive program to learn concepts. Besides GAs, popular inductive learning methods include neural networks (Rumelhart & McClelland, 1986) and decision trees (Quinlan, 1986, 1993).

In *incremental learning,* data becomes available over time, and the learning system must adapt what it has learned to the most recent data items. In *batch learning,* on the other hand, all data is available at the outset, and the learner is allowed to form a concept by digesting all data simultaneously. This article will concentrate exclusively upon batch learning.

In *unsupervised learning,* outcomes for situations represented by individual data items are not known. In *supervised learning,* outcomes for all data items from which the system is to learn are provided to the system. This article will concern itself exclusively with supervised learning.

Our generalized inductive learning problem can be summarized as follows. Given a database of examples of one or more concepts, return a general description of the concept(s) embodying those examples. The description should be sufficiently general to apply to both examples from the database, whose outcomes are all known, and examples outside the database, whose outcomes are unknown. We will follow Rendell's (1990) description of the general induction problem in much of what follows.

Since our goal is to learn "concepts," we must first define a concept. Rendell defines a concept as a "function that may have either certain or probabilistic values." The function maps *examples* (also called *instances*) to *classes* (also known as *categories* or *outcomes*). The task that any inductive learning system must accomplish is to return a function that performs an appropriate mapping.

Each example or instance in the database consists of a number of *attributes* (also called *variables* or *features*) and their associated values, along with the class to which each example belongs. In the general case, both attributes and classes can be of several types, such as boolean, integer, real, nominal (e.g., {red, green, blue}), or

hierarchical. Since our domain of application is financial, we will restrict our attention to real-valued attributes and either binary or real-valued classes. Therefore, an example in our case will be a vector of v real-valued variables, and will have associated with it either a binary or real-valued outcome. The variables will represent various factors, such as company earnings, that one might reasonably expect to have predictive value for the future performance of a stock. The classes will represent the performance of a particular stock, k time steps into the future. In some cases the classes will be binary-valued, representing buy and sell decisions based upon whether a stock outperforms or underperforms others. In other cases, the classes will be real-valued, representing the relative return of a stock.

The v variables of our problem define the axes of an *instance space.* Each example is a point in the instance space. The concept that is learned is a function over the instance space. For example, consider the one-dimensional instance space of Figure 1, defined by the variable x. Four examples are present in the database, two from Class 0 and two from Class 2. The figure shows two possible concepts that are completely consistent with the four examples. The first concept is a discontinuous step function, representing binary classification, and can be summarized by the following rule:

$$2 \leq x \leq 3 \Rightarrow \text{Class } 2 \qquad \text{otherwise, Class } 0$$

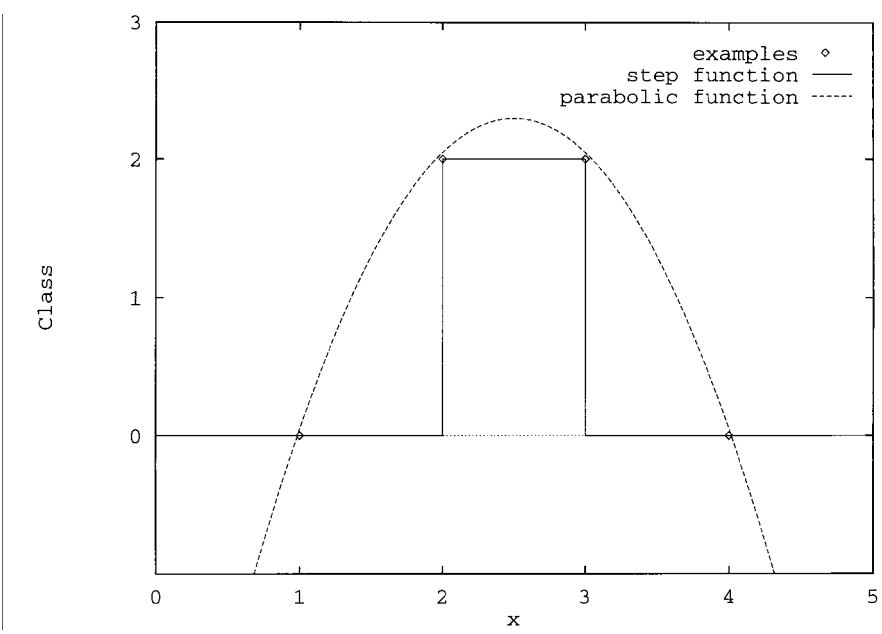

Figure 1. Concepts defined over a one-dimensional instance space.

The second concept is parabolic, representing fuzzy or continuous classification, and can be summarized by an appropriate interpolating or approximating parabola. For binary-valued financial predictions, a function similar to the step function in Figure 1 is appropriate. For real-valued financial predictions, the latter type of fuzzy or continuous concept is appropriate.

Of course, in the financial forecasting domain, exact concepts rarely exist. Concepts are typically nonlinear and also very noisy. In many cases, similar situations lead to different outcomes. The best one can do is to learn a concept that statistically produces good predictions more often than not.

Consider an example having only two variables, each of which can take on any integer value from 1 to 10. The corresponding two-dimensional instance space contains only 100 points. Suppose our objective is to generate buy or sell recommendations using the values of the two variables. Our concept is a function of two variables that outputs one of two classifications, one representing "buy" and another representing "sell." The number of possible concepts or *hypotheses* is 2^{100}, or over 10^{30}, equal to the number of possible functions, which is equal to the number of subsets that can be formed from the 100 possible instances. Clearly, to successfully search for any nontrivial function requires some method of limiting the number of hypotheses.

Biases (Utgoff, 1986) focus a learning method's search, forcing it to consider only hypotheses of a particular form. For instance, a neural network is itself a structural bias, upon which algorithms such as backpropagation operate. Decision rules are also structural biases. Biases transform the problem of learning a function defined over instance space to that of optimizing another function defined over hypothesis space (Rendell, 1990). (Optimization is the traditional GA's native mode of operation.) For instance, in the problem of Figure 1, if we bias our target concept to be a parabola, the problem reduces to finding the parabola that best fits our four examples. Since parabolas are of the form

$$f(x) = ax^2 + bx + c$$

our hypothesis space consists of an objective function defined over a three-variable space. (The variables to be optimized, namely, *a, b,* and *c,* each define an axis.) The objective function can simply be the sum of squared error terms between f values and database outcomes for each example, much like that used in the backpropagation algorithm for neural networks (Rumelhart et al., 1986). A problem that was previously unwieldy has now reduced to a parameter optimization problem.

GENETIC ALGORITHMS

GAs are general-purpose, parallel search techniques for solving complex problems. Based upon genetic and evolutionary principles, GAs work by repeatedly

modifying a population of artificial structures through the application of genetic operators. GAs require only fitness information, not gradient information or other internal knowledge of a problem. GAs have traditionally been used in optimization but, with a few enhancements, can perform classification and prediction as well.

GAs offer several advantages over traditional parameter optimization techniques. Given a nondifferentiable or otherwise ill-behaved problem, many traditional optimization techniques are of no use. Since GAs do not require gradient information, they can be used in such situations. In addition, GAs are designed to search highly nonlinear spaces for global optima. While traditional optimization techniques are likely to converge to a local optimum once they are in its vicinity, GAs conduct search from many points simultaneously, and are therefore more likely to find a global optimum. A further advantage is that GAs are inherently parallel algorithms, meaning that their implementation on multiple machines or parallel machines is straightforwardly accomplished by dividing the population among the available processors. Finally, GAs are adaptive algorithms (Holland, 1992), capable, in theory, of perpetual innovation.

The data structure upon which a GA operates can take a variety of forms. The choice of an appropriate structure for a particular problem is a major factor in determining a GA's success. Structures utilized in prior research include binary strings (Goldberg, 1989), computer programs (Koza, 1992), neural networks (Whitley et al., 1990), and if-then rules (Bauer, 1994).

We first examine how a traditional GA performs optimization. In optimization, the goal is ideally to find the *best* possible solution to a problem. For real-world problems, one does not usually know the best possible solution. Therefore, a more realistic objective is to find a *good* solution or, given a current benchmark, to search for a *better* solution. A GA's *fitness function* measures the quality of any particular solution.

The traditional GA begins with a population of n randomly generated structures, where each structure encodes a solution to the task at hand. The GA proceeds for a fixed number of *generations,* or until it satisfies some stopping criterion. During each generation, the GA improves the structures in its current population by performing selection, followed by crossover, followed by mutation. After a number of generations, the GA converges, meaning that all structures in the population become identical or nearly identical. The user typically chooses the best structure of the last population as the final solution.

Selection is the population improvement or "survival of the fittest" operator. Basically, it duplicates structures with higher fitnesses and deletes structures with lower fitnesses. A common selection method, called *binary tournament selection* (Goldberg & Deb, 1991), randomly chooses two structures from the population and holds a tournament, advancing the fitter structure to the crossover stage. A total of n such tournaments are held to fill the input population of the crossover stage.

Crossover, when combined with selection, results in good components of good structures combining to yield even better structures. Crossover forms $n/2$ pairs, randomly without replacement, from the n elements of its input population. Each pair advances two offspring structures to the mutation stage. The offspring are the results of cutting and splicing the parent structures at various randomly selected crossover points. The crossover stage advances a total of n elements to the mutation stage.

Mutation creates new structures that are similar to current structures. With a small, prespecified probability, mutation randomly alters each component of each structure. The mutation stage advances n elements to the selection stage of the next generation, completing the cycle.

Figure 2 illustrates one generation of a GA with a population of size $n = 4$. Rectangles represent structures, and arrows represent operations on structures. Each structure contains two square components. Components may take on one of two values, represented by black and white. Assume that structures with two black components have the highest fitness, structures with two white components have the lowest fitness, and mixed structures have intermediate fitness.

The selection stage holds four tournaments between randomly chosen pairs of individuals. In this example, Structures 1 and 2 compete (Structure 2 wins), as do Structures 1 and 4 (1 wins by tie-break), Structures 2 and 3 (3 wins by tie-break), and Structures 3 and 4 (3 wins). The crossover stage then cuts and splices structures at component boundaries. In this example, Structures 5 and 6 cross to form 9 and 10; Structures 7 and 8 cross to form 11 and 12. Finally, the mutation stage, through random choices, mutates only the leftmost component of Structure 9, yielding Structure 13. Although the initial population has no optimal structures, after the generation shown, an optimal structure emerges (Structure 14).

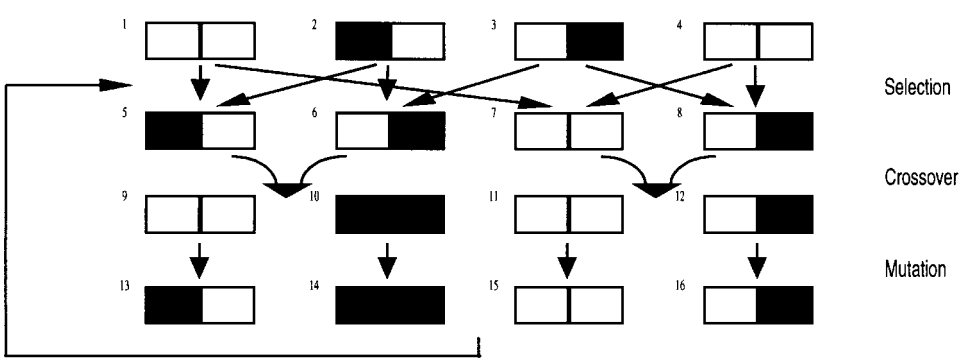

Figure 2. One generation of a genetic algorithm, consisting of—from top to bottom—selection, crossover, and mutation stages.

Genetic algorithms derive most of their power from crossover and from the simultaneous testing of many structural components. Crossover, in combination with survival of the fittest structures, allows the best components of differing solutions to combine to form even better solutions. Although GAs visit a fixed number of structures each generation, they effectively sample a much greater number of structural components, leading to a highly effective search. It is essentially these components—the genetic material—that the GA implicitly processes, and not individual solutions. For example, in a financial application, a GA may discover that one variable or component, such as price-to-earnings ratio, is a better indicator of future return than another variable, such as past return. In this case, the GA would place more emphasis on the price/earnings variable and less on the past-return variable.

GENETIC ALGORITHMS FOR INDUCTIVE LEARNING

There exist many components in the design of a GA-based inductive learning system. This section discusses the major components and reviews the choices made in previous GA-based systems and in the GA-based system of this article. This section also reviews prior applications of GAs in financial prediction.

Concept Description Structure

The choice of concept description structure is perhaps the strongest bias built into any GA-based inductive learning system. It will hence be a major factor determining how well our system works. The performance of any structure will depend partly upon other design parameters and even upon the problem being solved. Note that the GA system must have an internal representation of the structure (discussed in the section entitled Other Components) as well as an external representation that corresponds to the user's point of view.

GAs are capable, in principle, of optimizing any classification structure or set of structures. In fact, people have tried optimizing most traditional machine-learning structures as well as some nontraditional structures using GAs. These structures have ranged from neural network weights and topologies (Gruau & Whitley, 1993; Whitley et al., 1990, 1991, 1993; Whitley & Schaffer, 1992), to LISP programs (Koza, 1992), to regions of the instance space—similar to decision trees—induced by a splitting algorithm (Rendell, 1983, 1985; Sikora & Shaw, 1994), to expert-system rules (Montana, 1990), to weights for a game's evaluation function (Rendell, 1990), to weights and orientations for the *k*-nearest neighbor algorithm (Kelly & Davis, 1991; Punch et al., 1993), to finite-state automata (Fogel et al., 1966) and context-free grammars (Wyard, 1991), to production-system-like rules (Booker et al., 1989; De Jong et al., 1993; Greene & Smith, 1993, 1994; Holland, 1986; Janikow, 1993).

Higher-level constructs such as neural networks and LISP programs are very powerful representations. However, this power comes at the expense of an additional layer of complexity. Neural networks require search for an optimal topology as well as an optimal set of weights. If the user fixes the topology in advance, there is no guarantee that it will be a good one. LISP programs, similarly, require the user to supply a set of functions that are likely to be of use. Each additional function increases the branching factor of the search by one. Also, the user must find the right middle ground between supplying functions of too high a level and functions of too low a level. Both neural networks and LISP programs are tree structures, which require modification of the GA to accommodate variable-length structures and substructures, and which have traditionally been encoded at a higher level than that to which the traditional GA is suited.

We choose to optimize classification rules in our system because many rule syntaxes exist that can be directly manipulated by the GA and that are capable of representing any concept over an instance space. Classification rules also have the advantages of simplicity and general-purpose applicability.

Pittsburgh Versus Michigan Approach

There have historically been two approaches to genetic classification, named after the universities at which the approaches originated: the Michigan approach (Booker et al., 1989; Holland, 1986) and the Pittsburgh approach (De Jong et al., 1993; Janikow, 1993; Smith, 1980, 1983). The main property distinguishing the two approaches is whether each population element represents a single classification rule or a set of rules. Although the two approaches have come with other accessories, we will use this single property to define and distinguish them.

In the Michigan approach, each population element encodes a single rule, and the population as a whole acts as the concept. The main advantages of the Michigan approach are its much smaller memory requirements and faster processing time. Potential drawbacks of the Michigan approach are that mechanisms must be designed to maintain a cooperating and diverse set of rules within the population, to handle credit assignment (decide how much a single rule contributes to overall population performance), and to perform conflict resolution (decide which rule receives priority in case more than one matches an example). However, there exist mechanisms native to GAs, such as niching and decomposable fitness functions, that are capable of handling these challenges.

In the Pittsburgh approach, each population element is an entire concept, and the best population element at the end of the GA's run is the final concept used for classification. The Pittsburgh approach offers the advantages of simplified credit assignment (each population element can be assigned fitness based on how well it classifies the entire training set), easier conflict resolution, and less reliance on niching or other population diversification mechanisms. Its drawbacks are redundancy and the associated increased processing time.

From the standpoint of the Michigan approach, the Pittsburgh approach processes a population of populations or rule sets. Since optimizing a population of populations places a large handicap on a GA-based learner, and since the problems presented by the Michigan approach can be handled quite well by a GA, we choose the Michigan approach for our GA learning system.

Niching Method

When genetic algorithms are used for optimization, the goal is typically to return a single value, the best solution found to date. In fact, if the traditional GA is run for enough generations, the entire population ultimately converges to the neighborhood of a single solution (Goldberg & Segrest, 1987), even if a problem has multiple solutions of equivalent fitness. Unfortunately, a single rule is usually insufficient to represent the desired concepts in either the financial domain or other complex domains. A set of interacting rules is required.

GAs that employ *niching methods* (Mahfoud, 1992, 1995a, 1995b) are capable of finding and maintaining multiple rules using a single population. The basic idea behind niching is to simultaneously optimize multiple regions of the search space by reducing competition among sufficiently dissimilar individuals. In financial prediction, different rules within a single GA population can perform forecasting under different market environments and company-specific conditions.

Having chosen the Michigan approach, we assure that the population maintains a diverse and cooperating set of rules by incorporating a niching method. Previous applications of the Michigan approach have glossed over this part of the design but have usually built some sort of implicit niching into the GA. In many studies, the poor performance of the genetic system could be attributed to a poor method of niching.

Fitness sharing (Goldberg & Richardson, 1987) is one type of niching method that has been applied to classification (Booker, 1982; Horn et al., 1994; Kargupta & Smith, 1991; Packard, 1990; Smith & Valenzuela-Rendón, 1989). Fitness sharing works by reducing the fitnesses of similar population elements.

Crowding (De Jong, 1975; Mahfoud, 1992, 1994, 1995a) is another type of niching method that has also been applied to classification (Booker, 1982; Goldberg, 1983; Holland & Reitman, 1978; Sedbrook et al., 1991; Stadnyk, 1987). Crowding forces newly generated population elements to replace older elements that are similar.

Sequential niching (Beasley et al., 1993) is a third approach, implemented by Sikora and Shaw (1994) in their genetic classification system. Sequential niching repeatedly runs a traditional GA, each time making sure that the population searches a new area of the space.

Two previous learning systems that utilize implicit niching methods also deserve mention. In both systems (Greene & Smith, 1993, 1994; Schaffer, 1984, 1985), the

fitness function is decomposed into independent components, and different population elements are assigned to optimize each component.

Fitness Function: Credit Assignment

Most research on classification by GA has been for incremental rather than batch learning. In incremental systems, it is necessary to assign credit to rules over time. Given sufficient iterations of a credit assignment scheme, the fitnesses of a fixed population of rules should approach a steady state. Although results from incremental, stimulus-response GA classifiers are not directly applicable to batch learning, we review components of fitness, strength, payoff, or credit assignment schemes that may be applicable.

Wilson (1987) incorporates accuracy and generality in his payoff scheme. He mentions a trade-off between the two components and tries different combinations, including an adaptive parameter to control generalization. On boolean function learning tasks, he achieves his best result using a strong bias toward generality and a substantial penalty for making errors.

Booker (1982, 1985) gives credit for partial matches or "near misses," setting the final strength of a rule equal to its preliminary strength times a match score. Booker (1985) assigns a match score of 1 for exact matches, and $(l-k)/(l^2)$ for inexact matches, where k is the number of mismatched bits and l is the number of bits in the rule's encoding. This places heavy emphasis on exact matches. Booker achieves his worst result using an all-or-nothing method—full credit for a perfect match, none for a mismatch. He achieves better results giving a point of credit for each fixed position (of the rule) that matches, and 3/4 of a point for each wildcard position (thus giving more specific rules higher point totals).

A handful of GA systems have attempted to learn in batch mode. De Jong et al. (1993) assign the fitness of a set of rules equal to the square of the percent of training examples the set correctly classifies (under the Pittsburgh system). Janikow (1993) defines a complex fitness function that incorporates completeness, consistency, and cost (also under the Pittsburgh system). Greene and Smith (1993, 1994) base fitness assignment on misclassification and entropy. They use information gain (Quinlan, 1986) as fitness, but adjust it for accuracy to prevent overly general rules. They also employ certain ad hoc elements in their fitness function. Sikora and Shaw (1994) assign each rule a fitness f according to the following function: $f(C) = [p/(p+n)](p/\mathrm{P})(1/n)$, where p is the number of positive examples the rule covers, n is the number of negative examples the rule covers, P is the total number of positive examples in the training set, and C is the rule or "condition." The term $p/(p+n)$ in the above equation serves as a measure of rule accuracy; p/P measures generality; and $1/n$ is a term that penalizes C for covering negative examples.

With niching and the implicit cooperation it brings in place, we need only assign fitness to each rule based upon how well it solves a portion of the overall classifi-

cation problem. Three general principles emerge from the above survey of fitness assignment methods: (1) award higher fitnesses to more accurate classification rules, (2) award higher fitnesses to more general classification rules, and (3) when doing boolean or exact concept learning, penalize heavily for covering incorrect examples. The basic components of a fitness function can include such quantities as the number of positive examples a rule correctly covers, the number of negative examples in the training set, and the length or complexity of a rule. Where outcomes are fuzzy or continuous, the magnitudes of the outcomes can also be utilized in the fitness function. Hence fitness assignment is a multicriteria optimization problem. One can either combine all criteria into a single fitness function, or search for a Pareto-optimal set of solutions to the multicriteria optimization problem (Fonseca & Fleming, 1995). We choose to combine all criteria into a single fitness function.

Conflict Resolution

Sometimes an example is covered by more than one rule. If all rules covering a particular example indicate the same classification, there is no problem—the example takes on that classification. However, when the rules covering a particular example indicate two or more classifications, a *conflict* occurs, and a conflict resolution scheme is typically called. Many systems can ignore conflict resolution because they learn only single concepts (e.g., De Jong et al., 1993). The learned concept covers the positive examples, and any uncovered instance space is assumed to be negative. The major drawback is that multiple-concept learning is only possible through repeated single-concept learning. Even in single-concept learning, no class exists to limit the expansion of the single class into overgeneral territory.

One possible conflict resolution scheme is to not resolve conflicts. The system simply outputs that no decision can be reached, or it outputs all possible classifications and possibly some information to help the user reach a decision. This is acceptable in many domains in which an action is not required for every example the system encounters. However, other domains require the system to make an educated guess for each example presented. Even if no guess is required, it might be nice to have some indication, even a probabilistic one, of which classifications are likely.

A second possible conflict resolution scheme is to make a random choice between classifications indicated by the overlapping rules. A third is to choose the most common of the conflicting classifications by sampling the training data. A fourth is to choose the output indicated by the majority of the conflicting rules, with an appropriate tiebreaker if necessary. A drawback of this fourth scheme is that identical or nearly identical rules can "gang up" on a better but less frequent rule. Another drawback is that further conflict resolution schemes are likely to be needed to break ties. Grefenstette (1989) addresses these problems by first selecting, for each possible classification, the fittest rule suggesting that classification, and then

making a probabilistic choice, weighted according to fitness, from among those previously chosen.

Other conflict resolution schemes can be designed based on fitness (for instance, letting the fittest rule win). Prior fitness-based schemes, however, are almost exclusively designed for incremental learners. Conflict resolution schemes can also be based upon generality. One approach is the *default hierarchy* method (Holland, 1992), in which the most specific matching rule wins. The idea behind default hierarchies is to promote evolution of rules to handle special cases. Alternatively, the most general rule could be allowed to win. This would probably lead to a few rules performing all of the classification, and would all but eliminate the utility of specialists.

Other Components

Many other components are necessary in a GA-based induction system. While these are minor considerations compared to the ones outlined previously, each can have a profound effect on the system.

The first such component is a rule representation. An internal rule representation for the GA must be powerful enough to encode the types of rules desired, and flexible enough to preserve the properties that make GAs effective. A potential advantage of GAs over other machine-learning methods is that GAs can easily support a mixture of symbolic and numeric representations, within a single string—no conversion is necessary. After one has decided upon an encoding for each attribute, the GA can process the resulting string, without regard to its underlying meaning. Rule representations from prior GA research studies are too numerous to cover in detail. The type of rule we use is similar to that recommended by Bauer (1994). His GA finds thresholds for one or more financial variables, above or below which a stock is considered attractive. For instance, if the GA's structure consists of two variables representing a particular stock's price and earnings per share (EPS), the final rule the GA returns might look like the following:

$$\textbf{IF } [\textit{Price} < 15 \textit{ and } \textit{EPS} > 1] \textbf{ THEN } \textit{Buy}$$

Numerical bounds, relations, conjunctions, disjunctions, and negations can all be encoded as bits. For instance, the four relations, $>$, $<$, $=$, and $\neq$, can be encoded using a two-bit substring, where 00 corresponds to $>$, 01 to $<$, 10 to $=$, and 11 to $\neq$.

A second component is the set of genetic operators (other than selection). We stick with the standard operators of single-point crossover and mutation. However, many other operators, both general purpose and specialized, are available.

A third component is some method of handling uncovered examples. When an example is not covered by any rule, but one still wishes to make a classification, a problem similar to a conflict arises. As before, one can choose a class at random

from the outputs in the training set. A slightly better method is to choose the most frequent class indicated by the training data. Smarter strategies use partial matching (Grefenstette, 1989; Liepins & Wang, 1991; Wilson, 1987). Greene and Smith (1993, 1994) avoid the problem of uncovered examples by covering each training example upon initialization; an elitist GA ensures that all newly generated populations also cover all training examples. The authors do not mention any strategy for classifying uncovered examples from outside the training set. Of course, systems that learn single concepts do not have the problem of uncovered examples; anything uncovered is in the negative class.

A fourth component is some method for handling unknown or missing attribute values. In many real-world domains, a system may have to make decisions based upon partial information. We handle missing information prior to invoking the GA. From the GA's perspective, all examples are complete.

A fifth component is a stopping criterion for the GA. The simplest solution, which we adopt, is to run for a fixed number of generations. The number of generations should be chosen sufficiently high to allow the GA to finish converging, with high probability. Another option is to measure convergence and to stop the GA when the population effectively stops improving.

THE APPLICATION: FORECASTING INDIVIDUAL STOCK PERFORMANCES

Given a collection of historical data pertaining to a stock, the task is to predict the future performance of that stock. Specifically, we predict the relative return of a stock 12 weeks or approximately one calendar quarter into the future. Relative returns are computed with respect to a particular index, a weighted average of returns for similar securities. For example, if IBM stock is up 5% after one quarter, and the S&P 500 index is up 3% over the same period, then IBM's relative return is +2% for that period (relative to the S&P 500 index).

We employ 15 proprietary attributes representing technical as well as fundamental information about each stock. Examples consist of the values of the 15 attributes at specific points in time, as well as the relative return for the stock over the subsequent 12-week time period. Examples overlap, in the sense that the 12-week time period is only partially realized when the subsequent example in time is sampled. We utilize between 200 and 600 examples, depending upon the experiment and the amount of data available for a particular stock.

The GA system's task is to operate upon all of the attributes to find favorable or unfavorable circumstances with respect to the output variable being predicted. The GA system evolves populations of up to 150 interacting rules. Each rule indicates, in terms of the 15 attributes, the particular conditions that produce various price behaviors relative to the market.

Complex applications such as predicting stock performance are often highly nonlinear tasks. This means, for example, that when a 10% increase in one variable results in a buy signal, a 20% increase in that same variable could result in a sell signal. Often, a single, simplistic rule is insufficient to model relationships among financial variables. Sometimes a combination of rules is required. For example, consider the following two trading rules, based upon fundamental analysis, that a GA could potentially evolve.

$$\textit{Rule}1 : \textbf{IF } [P/E > 30] \textbf{ THEN } \textit{Sell}$$

$$\textit{Rule}2 : \textbf{IF } [P/E < 40 \textit{ and Growth Rate} > 40\%] \textbf{ THEN } \textit{Buy}$$

Given only the price-to-earnings (P/E) ratio of a particular stock, a good rule of thumb is that stocks with too high a P/E—in this example, greater than 30—are unattractive. Rule 1 encodes this heuristic. However, if a stock is a high-growth stock, one may wish to make an exception to Rule 1. Rule 2 encodes such an exception. The interacting combination of a general rule (Rule 1) plus an exception (Rule 2) results in a better trading strategy than either rule alone.

PRELIMINARY EXPERIMENTS

As a preliminary set of experiments, we attempt to predict the return, relative to the market, of a MidCap stock randomly selected from the S&P 400. We call the chosen stock X. The results below are summarized from a previous paper (Mahfoud & Mani, 1995).

There are 331 examples present in our database of examples for stock X. We use 70% of the examples as a *training set* for the GA; these are used to calculate fitnesses. Another 20% of the examples serve as a *stopping set,* to decide which population is best. Finally, 10% of the examples are used to measure performance; these make up the *out-of-sample* or *test set.*

We perform 100 different experimental runs of the GA, allocating examples to the three sets randomly for each experiment. The results we report are for the 100 out-of-sample sets. The GA returns one of three predictions for each out-of-sample example: up, down, or no prediction. Averaged over the 100 out-of-sample sets, the GA correctly predicts the direction of stock X relative to the market 47.6% of the time, produces no prediction 45.8% of the time, and incorrectly predicts the direction relative to the market 6.6% of the time. Thus, over half of the time (47.6% + 6.6%), the GA makes a prediction. When it does make a prediction, the GA is correct 87.8% of the time. Note that it is possible to force the GA to make a prediction every time, using techniques outlined in prior sections. However, the no-prediction option allows the GA to indicate those times when a stock is nearly equally likely to move in either direction.

We apply a second performance measure, called the average magnitude score, that takes into account magnitude as well as direction. If the GA correctly predicts the relative direction of a test example, then it receives as a score the absolute value of the actual return of the stock relative to the market. For an incorrect prediction, the score is the absolute value of the actual relative return, negated. The score is averaged over all test examples and then over the 100 experiments. The GA achieves an average magnitude score of +10.2%. By contrast, random guessing produces an expected average magnitude score that is slightly negative. Another naive strategy, choosing the most common direction in the training set, yields a slightly positive score (of less than +2%).

A sample rule that the GA generates in one of the experiments is of the following form:

IF [*Earnings Surprise Expectation* > 10% *and Volatility* > 7% *and* . . .]

THEN *Prediction* = *Up*

Such rules can serve as approximate explanations of how the various technical and fundamental input factors relate to future, individual stock returns.

We perform the same set of 100 experiments using a neural network system (Mani & Barr, 1994) with one layer of hidden nodes. Each experiment involves training the NN with the backpropagation algorithm, using the same training, stopping, and test sets as in the corresponding GA experiment.

For each experiment, the NN makes no prediction when the squared correlation on the stopping set is less than 0.5. Note that this decision is made experiment by experiment rather than example by example (as in the GA). The NN makes no prediction in only 5 out of 100 experiments (5% of the time). The NN correctly predicts the direction of stock X relative to the market 79.2% of the time, and incorrectly predicts direction relative to the market 15.8% of the time. When it does make a prediction, the NN is correct 83.4% of the time. The NN achieves an average magnitude score of +9.2%.

In the above experiments, the NN makes many more predictions than does the GA. One way of being more selective with the NN is to allow a prediction only when that prediction is more than 0.5 standard deviations away from the mean prediction (across all test examples). This helps eliminate many near-zero or noisy predictions. Under this new methodology, the NN makes no prediction 48.6% of the time, a correct prediction 47.4% of the time, and an incorrect prediction 4% of the time. When it does make a prediction, the NN is correct 92.2% of the time, achieving an average magnitude score of +13.4%. The overall results of the preliminary experiments are summarized in Table 1.

At the 95% confidence level, all four directional accuracies and all four magnitude scores in Table 1 differ significantly from each other, with one exception: on directional accuracy, the GA and NN ($|\sigma| > 0.5$) strategies do not significantly differ.

Table 1. Summary of preliminary experiments

Algorithm	Directional accuracy (%)	Predictions made (%)	Magnitude score (%)
GA	87.8	54.2	10.2
NN	79.2	95.0	9.2
NN ($\|\sigma\| > 0.5$)	92.2	51.4	13.4
Naive strategy	60.0	100.0	2.0

While the preliminary experiments shed ample light on the classification accuracy of the GA and NN systems, they do not say as much about predictive ability (looking strictly into the future). Since examples are interspersed randomly throughout time, we obtain a combination of using past examples to forecast future returns, and future examples to forecast past returns. While the latter scenario is an indicator of the potential of a forecasting system, the former scenario will exclusively be the case when taking a system to production. Another significant limitation of the preliminary experiments is that they operate upon a single stock. Some stocks may be inherently more predictable than others.

MORE EXTENSIVE EXPERIMENTS

The second, more extensive set of experiments closely simulates the full system in production because it is taken from the full GA and NN systems, in production. Both GA and NN forecasts are made at the end of the week for three consecutive weeks. Each weekend, forecasts are made for over 1600 stocks. Twelve weeks after each forecast, the forecasts are compared with actual 12-week relative returns, and aggregate statistics are gathered. In all, roughly 5000 different situations are modeled using both GAs and NNs. For each stock, the out-of-sample set consists of a single example whose future is completely unknown when the prediction is made.

Note that the current experiments differ from "backtested" experiments and are therefore free of the biases of backtested experiments. One such bias is *survivorship bias,* in which stocks that go bankrupt or otherwise disappear are not modeled because data is often unavailable. Not modeling such stocks typically skews aggregate performances. We model all stocks for which sufficient data is available at the date of modeling. Since the date of modeling is the present and not the past, by definition the experiments are free of survivorship bias.

Another type of bias typically present in backtested experiments is *look-ahead bias.* Look-ahead bias involves the inadvertent use of information that is not publicly available until a later date—forecasting the future by seeing the future. Much commercially and publicly available financial data, despite claims to the contrary by its providers, is likely to contain some sort of look-ahead bias. This is not typically due to laziness or malicious intent on the part of the providers, but to a constant

revision of past figures. Governmental economic indicators are especially suspect (at least in the United States) because they are often revised many months into the future, but left time-stamped with the original date of issue. Corporate earnings are occasionally revised as well. Again, since our date of modeling is the present and not the past, no look-ahead is possible.

The second set of experiments differs from the first set, in that we predict both the magnitude and direction of the return of each stock, relative to the market. We measure the average realized relative return of the positive forecasts (forecasts of outperforming the market) as well as the average realized relative return of the negative forecasts (forecasts of underperforming the market). We also measure the spread between these averages. The higher the spread, the better the overall set of predictions. We average the performance measures across all three time points. Our market index is simply the average return of the 1600+ stocks that we model each week, equally weighted. Hence, the objective is to identify which of the 1600+ stocks will outperform the others and which will underperform. Tables 2 and 3 show the results for the GAs and the NNs. Summary results for each table are shown in boldface.

Tables 2 and 3 show that both the GAs and the NNs have proficiency on both the buy and sell sides. (At the 95% confidence level, all results labeled Mean in Tables 2 and 3 differ significantly from zero.) Overall, the GAs outperform the NNs in this set of experiments, having an average spread between positive and negative forecasts of 2.16%, versus a spread of 1.40% for the NNs. (This difference is also statistically significant.) The NNs, however, make more predictions than the GAs. Overall, the NNs make predictions for 99.9% of the stocks run, while the GAs make predictions for 73.0% of the stocks run.

While Tables 2 and 3 give general performance measures for positive and negative forecasts as a group, they do not take into account the magnitudes of the forecasts. When making real investment decisions as to which stocks to purchase, one will generally prefer stocks with higher positive forecasts than lower positive forecasts. In fact, the positive forecasts in Tables 2 and 3 contain many near-zero forecasts. One would typically treat near-zero forecasts as if they were zero forecasts and not purchase the underlying stocks. Tables 4 and 5 show the positive forecasts broken down by quintile, and the average relative return of each quintile. Typically, one would purchase stocks only in the top one or two quintiles. We concentrate on

Table 2. Genetic algorithm forecasts

	May 19, 1995	May 26, 1995	June 2, 1995	Mean
Number of stocks modeled	1619	1645	1671	1645
All positive forecasts (%)	+0.35	+0.68	+2.02	+1.01
All negative forecasts (%)	–1.10	–1.06	–1.30	–1.15
Spread (%)	1.45	1.74	3.32	**2.16**

Table 3. Neural network forecasts

	May 19, 1995	May 26, 1995	June 2, 1995	Mean
Number of stocks modeled	1619	1645	1671	1645
All positive forecasts (%)	+0.71	+0.48	+0.90	+0.70
All negative forecasts (%)	–0.72	–0.45	–0.93	–0.70
Spread (%)	1.43	0.93	1.83	**1.40**

purchases rather than sales because our inputs are currently designed more for proficiency on the buy side. Negative forecasts tend to have more uniform performances across quintiles.

In the top quintile, the GAs score an actual relative return of 5.47% versus 4.40% for the NNs. GAs and NNs perform roughly equivalently in the second quintile. The GAs show a smooth degradation of performance, on the average, down to the fourth quintile. The bottom quintile is somewhat of an anomaly, outperforming the third quintile. The NNs show a smooth degradation of performance, on the average, all the way down to the bottom quintile.

A prior paper (Mahfoud & Mani, 1995) raises the question of whether an approach that utilizes both GA and NN forecasts is superior to utilizing the better of the two algorithms, in isolation. The combined approach is analogous to having multiple human experts independently perform the same task and pool their results afterward. As a preliminary test, we combine forecasts from the GAs and the NNs. The end result is an average relative return of 6.61% in the top quintile of the positive combined forecasts. This is a 21% improvement over the GAs alone, and a 50% improvement over the NNs alone. Hence our preliminary results indicate that synergy does result from a combined approach.

Note that the previous relative return percentages represent the excess return over that produced by "the market," which in our case is the average return produced by the 1600+ stocks modeled. Table 6 compares the 12-week absolute returns produced by the top GA forecasts, the top NN forecasts, the top combined forecasts, the 1600+ stocks modeled, and three popular market indices—the S&P 500, which

Table 4. Actual relative returns for positive genetic algorithm forecasts, broken down by quintile

Quintile	May 19, 1995	May 26, 1995	June 2, 1995	Mean
Top (%)	+4.38	+6.33	+5.70	**+5.47**
Second (%)	+0.47	–0.04	+4.18	+1.53
Third (%)	–1.18	–1.19	–0.11	–0.83
Fourth (%)	–1.80	–1.03	–0.79	–1.21
Bottom (%)	–0.13	–0.65	+1.11	+0.11

Table 5. Actual relative returns for positive neural network forecasts, broken down by quintile

Quintile	May 19, 1995	May 26, 1995	June 2, 1995	Mean
Top (%)	+4.87	+5.29	+3.03	**+4.40**
Second (%)	+1.39	+0.87	+2.20	+1.49
Third (%)	–0.40	–1.11	+0.77	–0.25
Fourth (%)	–0.44	–0.40	–0.97	–0.60
Bottom (%)	–1.88	–2.26	–0.51	–1.55

represents stocks of large companies; the S&P 400, which represents stocks of midsized companies; and the Russell 2000, which represents stocks of small-sized companies. The 1600+ stocks represent companies of all sizes. All three technologies—GAs, NNs, and combined—significantly outperform all market indices, at the 95% statistical confidence level. It is also apparent from Table 6 that one could not have generated the results produced by the GAs and NNs by simply screening on the basis of a company's size.

DISCUSSION: GENETIC ALGORITHMS VERSUS NEURAL NETWORKS

The results demonstrate that genetic algorithms and neural networks are both promising methods for predicting individual stock performances. Their success in this study is due to their ability to learn nonlinear relationships, among the input variables, that result in a stock outperforming or underperforming the market. These nonlinear relationships are typically not apparent to the average market participant.

Overall, the GAs outperform the NNs on the chosen task. We attribute part of this outperformance to the fact that the GAs abstain from making predictions 27% of the time, while the NNs make predictions in nearly all cases. (NNs must explicitly be restricted in order to make fewer forecasts.) The GAs may thus have an advantage

Table 6. Top genetic algorithm and neural network predictions versus the market indices

	May 19, 1995	May 26, 1995	June 2, 1995	Mean
Top quintile combined (%)	15.37	18.47	13.69	**15.84**
Top quintile GAs (%)	12.93	16.26	14.91	**14.70**
Top quintile NNs (%)	13.42	15.22	12.24	**13.63**
1600^{+} stocks (%)	8.55	9.93	9.21	**9.23**
S&P 500 (large) (%)	6.92	6.79	5.18	**6.30**
S&P 400 (medium) (%)	9.44	11.14	9.63	**10.07**
Russell 2000 (small) (%)	10.55	12.47	11.79	**11.60**

in deciding when a stock is less predictable. We are currently investigating methods for enabling the GAs to make more predictions, without substantial loss of accuracy. We are also investigating methods of restricting the NNs to not make predictions in cases where a stock is not readily able to be modeled. In such cases, it is unlikely that any prediction will be better than a random guess. For various prediction methods, a trade-off exists between the number of predictions made and overall accuracy. This trade-off is a topic for future research.

NNs tend to do interpolation and approximation of data, while GAs tend to do clustering. This may account for the smooth degradation in the NN forecasts, as forecasted magnitude decreases, and the not-as-smooth degradation in the GA forecasts.

One advantage of GAs over NNs is the GA's ability to output comprehensible rules. This points to two possible applications of GAs, beyond prediction. One is to provide a rough explanation of the concepts learned by black-box approaches such as NNs. The other is to learn rules that are subsequently embedded in a formal expert system. The advantage of this automated method of building an expert system is that no expert is required, and hence the tedious process of transforming that expert's knowledge into a set of rules is eliminated. In addition, an expert is likely to produce a set of rules qualitatively different than that produced by a GA, due in part to the expert's a priori bias against counterintuitive or contrarian rules.

The GA and NN systems are evolving systems, both of which have been under development for years. We feel that both systems can be further improved, and are in the process of putting improvements into production every few weeks. The algorithm that is better today might not be better tomorrow.

The greatest remaining limitation of our experiments is the time frame over which they are run. One way to extend the time frame is to do backtesting over a several-year time period, choosing a variety of points in time under various market conditions. Of course, survivorship bias and look-ahead bias then become issues. We are in the process of performing such backtests. Another approach is to extend the type of simulation in this article as more production results become available. We are also taking this approach, benchmarking results as they come in.

CONCLUSION

This article has presented a new genetic-algorithm-based system for inductive machine learning. The system is applied to financial forecasting, specifically to predicting the performances of individual stocks. The genetic algorithm system is benchmarked against a neural network system on roughly 5000 stock-prediction experiments. In the experiments conducted, both systems significantly outperform "the market," with the genetic algorithm system producing better final results. A combined approach outperforms either algorithm individually.

REFERENCES

Bauer, R. J., Jr. 1994. *Genetic algorithms and investment strategies.* New York: John Wiley.

Beasley, D., D. R. Bull, and R. R. Martin. 1993. A sequential niche technique for multimodal function optimization. *Evolutionary Computation* 1(2):101–125.

Booker, L. B. 1982. Intelligent behavior as an adaptation to the task environment. *Dissertation Abstracts International* 43(2):469B. (University Microfilms No. 8214966.)

Booker, L. B. 1985. Improving the performance of genetic algorithms in classifier systems. In *Proceedings of an international conference on genetic algorithms and their applications,* 80–92.

Booker, L. B., D. E. Goldberg, and J. H. Holland. 1989. Classifier systems and genetic algorithms. *Artificial Intelligence* 40:235–282.

De Jong, K. A. 1975. An analysis of the behavior of a class of genetic adaptive systems. *Dissertation Abstracts International* 36(10):5140B. (University Microfilms No. 76-9381.)

De Jong, K. A., W. M. Spears, and D. F. Gordon. 1993. Using genetic algorithms for concept learning. *Machine Learning* 13(2/3):161–188.

Fogel, L. J., A. J. Owens, and M. J. Walsh. 1966. *Artificial intelligence through simulated evolution.* New York: John Wiley.

Fonseca, C. M., and P. J. Fleming. 1995. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation* 3(1):1–16.

Goldberg, D. E. 1983. Computer-aided gas pipeline optimization using genetic algorithms and rule learning. *Dissertation Abstracts International* 44(10):3174B. (University Microfilms No. 8402282.)

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

Goldberg, D. E., and K. Deb. 1991. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of genetic algorithms,* ed. G. J. E. Rawlins, 69–93. San Mateo, Calif.: Morgan Kaufmann.

Goldberg, D. E., and J. Richardson. 1987. Genetic algorithms with sharing for multimodal function optimization. In *Genetic algorithms and their applications: Proceedings of the second international conference on genetic algorithms,* 41–49.

Goldberg, D. E., and P. Segrest. 1987. Finite Markov chain analysis of genetic algorithms. In *Genetic algorithms and their applications: Proceedings of the second international conference on genetic algorithms,* 1–8.

Greene, D. P., and S. F. Smith. 1993. Competition-based induction of decision models from examples. *Machine Learning* 13(2/3):229–257.

Greene, D. P., and S. F. Smith. 1994. Using coverage as a model building constraint in learning classifier systems. *Evolutionary Computation* 2(1):67–91.

Grefenstette, J. J. 1989. A system for learning control strategies with genetic algorithms. In *Proceedings of the third international conference on genetic algorithms,* 183–190.

Gruau, F., and D. Whitley. 1993. Adding learning to the cellular development of neural networks: Evolution and the Baldwin effect. *Evolutionary Computation* 1(3):213–233.

Holland, J. H. 1986. Escaping brittleness: The possibilities of general purpose learning algorithms applied to parallel rule-based systems. In *Machine learning: An artificial intelligence approach,* vol. 2, eds. R. Michalski, J. Carbonell, and T. Mitchell. San Mateo, Calif.: Morgan Kaufmann.

Holland, J. H. 1992. *Adaptation in natural and artificial systems.* Cambridge, Mass.: MIT Press.

Holland, J. H., and J. S. Reitman. 1978. Cognitive systems based on adaptive algorithms. In *Pattern-directed inference systems,* eds. D. A. Waterman and F. Hayes-Roth, 313–329. New York: Academic Press.

Horn, J., D. E. Goldberg, and K. Deb. 1994. Implicit niching in a learning classifier system: Nature's way. *Evolutionary Computation* 2(1):37–66.

Janikow, C. Z. 1993. A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning* 13(2/3):189–228.

Kargupta, H., and R. E. Smith. 1991. System identification with evolving polynomial networks. In *Proceedings of the fourth international conference on genetic algorithms,* 370–376.

Kelly, J. D., Jr., and L. Davis. 1991. Hybridizing the genetic algorithm and the *k*-nearest neighbors classification algorithm. In *Proceedings of the fourth international conference on genetic algorithms,* 377–383.

Koza, J. R. 1992. *Genetic programming: On the programming of computers by means of natural selection.* Cambridge, Mass.: MIT Press.

Liepins, G. E., and L. Wang. 1991. Classifier system learning of boolean concepts. In *Proceedings of the fourth international conference on genetic algorithms,* 318–323.

Mahfoud, S. W. 1992. Crowding and preselection revisited. In *Parallel problem solving from nature,* 2, eds. R. Männer and B. Manderick, 27–36. Amsterdam: Elsevier.

Mahfoud, S. W. 1994. Crossover interactions among niches. In *Proceedings of the first IEEE conference on evolutionary computation, IEEE world congress on computational intelligence,* 188–193.

Mahfoud, S. W. 1995a. Niching methods for genetic algorithms. *Dissertation Abstracts International* 56(9):4987B. (University Microfilms No. 9543663.)

Mahfoud, S. W. 1995b. Population size and genetic drift in fitness sharing. In *Foundations of genetic algorithms,* vol. 3, eds. L. D. Whitley and M. D. Vose, 185–223. San Francisco, Calif.: Morgan Kaufmann.

Mahfoud, S., and G. Mani. 1995. Genetic algorithms for predicting individual stock performance. In *Proceedings of the third international conference on artificial intelligence applications on Wall Street,* 174–181. Gaithersburg, Md.: Software Engineering Press.

Mani, G., and D. Barr. 1994. Stock-specific, non-linear neural net models: The AXON system. Paper presented at the second international workshop on neural networks in the capital markets, Pasadena, Calif.

Michalski, R. S. 1983. A theory and methodology of inductive learning. In *Machine learning: An artificial intelligence approach,* eds. R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, 83–134. Palo Alto, Calif.: Tioga.

Montana, D. J. 1990. Learning thresholds for expert system rules. *Machine Learning* 5(4):427–450.

Packard, N. H. 1990. A genetic learning algorithm for the analysis of complex data. *Complex Systems* 4(5):543–572.

Punch, W. F., E. D. Goodman, M. Pei, C. S. Lai, P. Hovland, and R. Enbody. 1993. Further research on font selection and classification using genetic algorithms. In *Proceedings of the fifth international conference on genetic algorithms,* 216–222.

Quinlan, J. R. 1986. Induction of decision trees. *Machine Learning* 1(1):81–106.

Quinlan, J. R. 1993. *C4.5: Programs for machine learning.* San Mateo, Calif.: Morgan Kaufmann.

Rendell, L. R. 1983. A doubly layered, genetic penetrance learning system. In *Proceedings of the third national conference on artificial intelligence,* 343–347.

Rendell, L. R. 1985. Genetic plans and the probabilistic learning system: Synthesis and results. In *Proceedings of an international conference on genetic algorithms and their applications,* 60–73.

Rendell, L. R. 1990. Induction as optimization. *IEEE Transactions on Systems, Man, and Cybernetics* 20(2):326–338.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams. 1986. Learning internal representations by error propagation. In *Parallel distributed processing: Explorations in the microstructure of cognition: Volume I: Foundations,* eds. D. E. Rumelhart and J. L. McClelland, 318–362. Cambridge, Mass.: MIT Press.

Rumelhart, D. E., and J. L. McClelland (eds.). 1986. *Parallel distributed processing: Explorations in the microstructure of cognition: Volume I: Foundations.* Cambridge, Mass.: MIT Press.

Schaffer, J. D. 1984. Some experiments in machine learning using vector evaluated genetic algorithms. Doctoral dissertation. Vanderbilt University, Nashville, Tenn.

Schaffer, J. D. 1985. Multiple objective optimization with vector evaluated genetic algorithms. In *Proceedings of an international conference on genetic algorithms and their applications,* 93–100.

Sedbrook, T. A., H. Wright, and W. Wright. 1991. Application of a genetic classifier for patient triage. In *Proceedings of the fourth international conference on genetic algorithms,* 334–338.

Sikora, R., and M. J. Shaw. 1994. A double-layered learning approach to acquiring rules for classification: Integrating genetic algorithms with similarity-based learning. *ORSA Journal on Computing* 6(2):174–187.

Smith, R. E., and M. Valenzuela-Rendón. 1989. A study of rule set development in a learning classifier system. In *Proceedings of the third international conference on genetic algorithms,* 340–346.

Smith, S. F. 1980. A learning system based on genetic adaptive algorithms. Doctoral dissertation. University of Pittsburgh, Pittsburgh, Pa.

Smith, S. F. 1983. Flexible learning of problem solving heuristics via adaptive search. In *Proceedings of the eighth international joint conference on artificial intelligence,* 422–425.

Stadnyk, I. 1987. Schema recombination in a pattern recognition problem. In *Genetic algorithms and their applications: Proceedings of the second international conference on genetic algorithms,* 27–35.

Utgoff, P. E. 1986. Shift of bias for inductive concept learning. In *Machine learning: An artificial intelligence approach,* vol. II, eds. R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, 107–148. Los Altos, Calif.: Morgan Kaufmann.

Whitley, D., S. Dominic, and R. Das. 1991. Genetic reinforcement learning with multilayer neural networks. In *Proceedings of the fourth international conference on genetic algorithms,* 562–569.

Whitley, D., S. Dominic, R. Das, and C. W. Anderson. 1993. Genetic reinforcement learning for neurocontroller problems. *Machine Learning* 13(2/3):259–284.

Whitley, D., and J. D. Schaffer (eds.). 1992. *Combinations of genetic algorithms and neural networks.* Los Alamitos, Calif.: IEEE Computer Society Press.

Whitley, D., T. Starkweather, and C. Bogart. 1990. Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel Computing* 14:347–361.

Wilson, S. W. 1987. Classifier systems and the animat problem. *Machine Learning* 2(3):199–228.

Wyard, P. 1991. Context free grammar induction using genetic algorithms. In *Proceedings of the fourth international conference on genetic algorithms,* 514–518.

PAGE 566 BLANK