

An Investigation of Genetics-Based Machine Learning as Applied to Global Crop Yields

Will Gantt

April 30, 2017

1 Introduction

As Earth’s population continues to grow and as the effects of climate change intensify, it is critical that we better understand the environmental and economic factors that most significantly affect agricultural output around the world. While rising temperatures may benefit certain crops in the short term, extreme weather and higher levels of atmospheric CO₂ will, if left unchecked, have disastrous consequences for water supply, soil fertility, and, consequently, for crop yields. [1]. Moreover, the United Nations Department of Economic and Social Affairs predicts that the world population will reach 9.7 billion by 2050 and 11.2 billion by the end of the century [2].

These forecasts demand prudent planning, which in turn requires that we be able to discern trends in agricultural data. To that end, I have designed and written a learning classifier system (LCS)—a powerful and versatile tool for data mining—and have applied it to data collected by Erik Nelson in order to probe the relationships between various agricultural inputs and changes in crop yields. This paper presents the preliminary results of my investigations.

The following section gives a brief overview of genetics-based machine learning (GBML) and LCSs, and of the research done by Nelson and Congdon that gave rise to this project [3]. Section 3 provides a thorough description of the system design of my own LCS. Section 4 details the experiments conducted and the results. Finally, section 5 offers some concluding remarks and suggests directions for future work, and section 6 contains experimental data.

2 Background

In this section, I provide some context for my project. First, I introduce the GBML paradigm and consider the place of LCSs within it. Next, I give a general overview of LCSs and their basic structure. I then discuss a few important developments in the history of LCSs for supervised learning, and conclude by talking specifically about the earlier work of Nelson and Congdon [3].

2.1 Genetics-Based Machine Learning

GBML is a machine learning approach based on the use of evolutionary computation. Briefly, evolutionary computation comprises a set of optimization algorithms that evolve populations of candidate solutions to a particular

problem through a process inspired by biological evolution. Many of the most interesting and important problems in machine learning have large and noisy search spaces, and it is in such contexts that techniques from evolutionary computation have proved particularly effective.

The precise relationships between various GBML-related terms and concepts have changed over the years, and usage in the community is often loose. GBML, once taken to refer exclusively to LCSs, is now considered a broader term that encompasses a number of other algorithms, including genetic programming, evolutionary ensembles, evolutionary neural networks, and genetic fuzzy systems [4]. For the purposes of this paper, it is important only to recognize that LCSs exist within a larger machine learning problem-solving framework.

2.2 What is a Learning Classifier System?

2.2.1 Structure

Introduced by John Holland in 1975, LCSs are GBML algorithms that evolve a population of rules¹ [5]. Each rule consists of a condition that indicates when the rule applies, as well as an action to take if the condition’s criteria are met. As LCSs have applications to a variety of learning problems, the function of the rules depends largely on the task. With supervised learning problems, rules attempt to categorize a set of training examples by mapping the features of those examples to particular classes. In these cases, a rule’s condition describes the features of the examples to which it applies, and the action specifies the class to be assigned to examples matching the condition. Problems in reinforcement learning, by contrast, often involve determining the best action for an agent to take in response to certain inputs or stimuli from the environment. Here, a rule’s condition describes a set of inputs and its action determines how the agent should react given those inputs.

Regardless of the types of problems to which they are applied, all LCSs share certain basic structures and mechanisms. Holmes et al. have identified four such components [6]. However, their model is more characteristic of LCSs used in reinforcement learning than those used in supervised learning. I suggest a slight modification of that model that covers both:

1. *Population.* All LCSs have a population of rules. Although some systems allow for the growth or reduction of the population, there is

¹Rules are also frequently referred to as “classifiers.” To maintain a clear distinction between individual *classifiers* and the *classifier systems* that comprise them, I use the term “rule” exclusively throughout to denote the former, and “LCS” to denote the latter.

typically a limit on the maximum number of rules it may contain.

2. *Learning.* An LCS must have a means of coercing its population toward rules that generate good actions or accurate classifications. Numerous methods have been used for this purpose and they tend to vary based on the kind of learning. Many early LCSs designed for reinforcement learning used the bucket brigade algorithm [7], while more contemporary systems often rely on Q-learning [8, 9] or Q-learning-inspired algorithms. For supervised learning tasks, other systems have used genetic algorithms exclusively [10]. Still others have used different approaches, including ensemble learning [11] and bayesian methods [12].
3. *Discovery.* An LCS must also have mechanisms for generating new rules. For that purpose, a vast majority of systems employ genetic algorithms (GAs), which provide two useful operations: crossover and mutation. Crossover produces two “offspring” rules from two existing “parent” rules, and mutation alters some of the attribute values of the offspring. However, many LCSs employ additional operators. The *covering* operator, for example, which creates a new rule when there are none that match the current input, has been widely used [9, 13, 14]. *Specify*, a second popular operator, proposed by Pier Lanzi, attempts to counteract the over-generalization of rules by specifying the attribute values of a rule according to a particular input [15].
4. *Classification or Action Selection.* Finally, an LCS must have some procedure for deciding what action to take or how to classify a particular example. In the typical case, the action or classification is selected based on the actions recommended by the rules in the *match set*—the set of rules matching a particular input. One possibility is simply to choose the action of the fittest rule in the match set. Another is to conduct a weighted vote of all the rules in the set, where each rule casts a “vote” for its action, weighted by its fitness. The selected action, then, is the one receiving the greatest number of points. Some systems have also incorporated random action selection alongside one of the first two methods [13].

These are, at the broadest level, the modules that feature in all LCSs. There is, of course, substantial variation between implementations, which is to be expected of a paradigm with applications as diverse as those of the LCS algorithm.

2.2.2 Genetic Algorithms

As GAs constitute such a critical component of LCSs, they merit at least a short and very general overview. GAs evolve populations of candidate solutions to a particular problem by a process inspired by biological evolution. In a traditional implementation, individuals are represented as bit strings, where each index in the strings corresponds to a particular property of the solution and a 1 or 0 at that index indicates the presence or absence of the property, respectively. In LCSs, a third character (a “don’t care” value, often denoted by “#”) is often used to indicate that the property is not relevant for a given rule. In principle, solutions may take any form, and GA variants have been developed to accommodate structures of different kinds, including objects [16]. The canonical GA works as follows²:

1. *Initialization.* Generate a random population of candidate solutions.
2. *Evaluation.* Evaluate the fitness of each individual according to some fitness function.
3. *Selection.* Select n individuals from the population to reproduce. The probability that a given individual will be selected is usually proportional to its fitness.
4. *Crossover.* Randomly pair the n selected individuals. For each pair, combine the bits of one member of the pair with those of the other (e.g. via one-point crossover) to produce two new solutions (“offspring”).
5. *Mutation.* Flip some number of the bits of each of the offspring with probability p .
6. *Replacement.* Replace the n least-fit members of the original population with the offspring created in (5).
7. Repeat (2)-(6) until the stopping condition (e.g. a target average fitness, a specific number of generations, or a runtime limit) is met.

The structure of the GA is just one respect in which different implementations of LCSs can differ. The next section outlines a second.

²For ease of illustration, I have used bit strings as the solution type in this example.

2.2.3 Pittsburgh and Michigan Styles

One of the most common and useful categorizations of LCSs distinguishes between *Michigan*- and *Pittsburgh*- or *Pitt-style* systems. The difference pertains to what counts as an “individual” in the population. In Michigan systems, an individual is a single rule. In Pitt systems, however, an individual is an entire rule *set*. Correspondingly, a solution in a Michigan system often comprises the entire population, while a solution in a Pitt system consists of a single individual (a rule set). This difference has important implications for many aspects of LCS design, including the learning algorithm, credit assignment, rule syntax, and genetic operators. Perhaps unsurprisingly, each style has notable advantages and disadvantages with respect to the other. Michigan systems, for example, typically require less memory and are less computationally intensive, while Pitt systems avoid the difficulties involved in distributing credit across individual rules. Neither type has emerged as obviously superior to the other, but Michigan systems do appear to predominate in the literature [17].

2.3 Developments in LCS Research

Numerous thorough histories and comparisons of LCS research have been written over the years [17, 18, 19, 20], and it is beyond the scope of this paper to undertake another one. Instead, I wish to highlight just a few key milestones in the particular domain of LCS research with which my project is concerned—namely, supervised learning and data mining. Each of the systems listed introduced at least one important concept to that domain.

2.3.1 LS-2

Schaffer and Grefenstette’s LS-2 [21] was the first attempt to use an LCS for a genuine supervised learning classification task. Specifically, they applied their system to the multi-class problem of classifying five human gait types based on EMG signals from the leg muscles. A total of 11 training examples were used, and rules were represented as 12-bit strings from the ternary alphabet $\{0, 1, \#\}$, with each index representing a particular signal.

LS-2 evolved a population of rule *sets* and may thus anachronistically be categorized as a Pitt-style LCS, though the distinction had yet to be formulated when the article was published. Of particular note in this system is the fitness scheme. LS-2 represented fitnesses, not as scalar quantities, but as vectors whose elements corresponded to the different possible classes (gait types). For each rule set, a fitness vector was computed as a function of the

number of correctly and incorrectly classified examples in each class. When constructing the population for the next generation, a portion of the rules were selected on the basis of each element in the fitness vector. In allowing fitness to be relativized to individual classes, Schaffer and Grefenstette introduced the notion of *niching*—the specialization of rules to particular regions of the solution space—which featured heavily in later systems, and which remains a critical topic in contemporary research.

2.3.2 FCS

Many kinds of data in the real world have continuous-valued attributes. Given the traditional bit string rule representation of an LCS, determining the best way to handle data of this sort is a challenge. In 1991, Manuel Valenzuela-Rendón proposed a solution to this problem using fuzzy sets [22]. In fuzzy set theory, the membership of an element in a set is determined by a membership function. Defining membership in this way allows for elements to be *partial* members of sets.

Valenzuela-Rendón’s “fuzzy classifier system” (FCS) applied this logic to rule representation. Suppose the condition of a rule R contains a “temperature” attribute, whose possible values are “cold,” “mild,” and “hot.” Instead of assigning R one of those values, FCS would define a membership function for each value, and R would be assigned some combination of those functions. If, for example, R was assigned the functions for “cold” and “mild,” the result would be a curve for the temperature attribute that, given a real-valued temperature, indicated the *degree* to which that temperature should be considered “cold” or “mild.”

In short, structuring rules in this way introduced two valuable concepts to the LCS community: (1) the ability of LCSs to handle real-valued inputs, and (2) the possibility of eliminating hard boundaries between classes.

2.3.3 XCS

Although designed explicitly for reinforcement learning, the “eXtended Classifier System” (XCS), proposed by Wilson in 1995, exerted such a profound influence on LCS research of all kinds that it deserves mention here [13]. Nearly all of the key features of XCS had been implemented in earlier systems, but its impressive accuracy and generalization capabilities made clear for the first time the potential effectiveness of those features.

XCS pioneered a slew of techniques that have since become staples of LCS design, particularly in reinforcement learning, including the use of

Watkins’s Q-learning algorithm for credit assignment and the use of a niche GA [17]. However the technique of greatest relevance to supervised learning was the use of accuracy-based fitness. Wilson was not the first to use accuracy-based fitness (the credit goes to Holland and his first LCS, CS-1 [23]), but a substantial majority of previous systems used a different parameter known as *strength*. Strength was decidedly a reinforcement learning mechanism, and measured the expected reward (or “reward prediction”) from the environment if the action of a particular rule was taken. The problem with strength-based fitness was that it tended to eliminate rules that predicted lesser rewards, but that nonetheless reliably led to the best action in specific circumstances. Even though accuracy had been used before, XCS radically increased its popularity and is now considered the primogenitor of all contemporary accuracy-based systems [17].

2.3.4 UCS

In 2003, Bernadó-Mansilla and Garrell-Guiu presented a supervised learning variation on XCS [14]. Dubbed “UCS,” their system adapted the reinforcement learning model of XCS to better suit multi-objective classification tasks. UCS retained all of the main features of XCS, including its accuracy-based fitness scheme, a GA that encouraged niching, and an online learning style, but it changed the manner in which accuracy was computed. The details of the differences are not salient; rather, the significance of UCS consists in its demonstration that ideas from reinforcement learning within LCS research could, with some effort, be refashioned to yield good solutions to problems in supervised learning.

2.4 Global Crop Yields

The remainder of this section summarizes the research conducted by Erik Nelson and Clare Bates Congdon in [3]. Their work — and mine — examines data on crop yields collected by Nelson and his students in the economics department. The data relate various environmental and economic variables to the agricultural output of countries, both in units of mass (Mg/ha) and in units of energy (Mkcal/ha), from 1975 to 2007. The variables considered cover such categories as growing season weather; crop choice; investment in irrigation capability, land, and machinery; agricultural technology; fertilizer use; and cropped footprint.

Using these data, Nelson and Congdon evaluated the relative impact of the year-to-year changes in the different variables on the change in global and

Variable	Description
\mathbf{X}_{ct}	a vector of harvested hectare percentages across different crops.
\mathbf{K}_{ct}	a vector of variables measuring investment in agricultural machinery and equipment per harvested hectare.
A_{ct}	the total harvested hectarage.
S_{ct}	the soil quality.
I_{ct}	the percentage of harvested land equipped for irrigation.
\mathbf{Z}_{ct}	a vector of various annual weather statistics.
F_{ct}	the total amount of fertilizer used.

Table 1: Variables used in Nelson and Congdon’s analysis of crop yield data [3]

regional crop yields [3]. The authors applied two analytical methods: fixed-effects econometric modeling and decision trees, each of which is explained in greater depth below. For both, annual changes in yield were discretized to the categories “high” (H), “medium” (M), and “low” (L).

2.4.1 Fixed Effects Modeling

Fixed-effects models offer a means of accounting for any unobserved, time-independent factors that affect a dependent variable in the analysis of panel data—that is, multi-dimensional data containing measurements across time. In Nelson and Congdon’s work, the dependent variables considered were the per-hectare crop yield in metric tons (Mg) and in millions of kilocalories (Mkcal). Let y_{ct} be the per-hectare crop yield (in whichever units) for a country c in a year t . The fixed effects model, then, has the form

$$y_{ct} = \beta_1 x_{ct1} + \beta_2 x_{ct2} + \dots + \beta_N x_{ctN} + \mu_{ct} + a_c$$

Where x_{cti} is the value of the independent variable x_i for country c in year t ; β_i is the coefficient for the independent variable x_i ; μ_{ct} (the “idiosyncratic error”) covers all unobserved, time-variant effects on y_{ct} ; and a_c (the “fixed effect”) captures all unobserved and time-invariant effects. The specific variables used are shown in table 1 (as above, the subscript “ ct ” indicates that the variable is for a country c in a year t).

Thus, their fixed effects model, computed using the method of least squares, has the form

$$Y_{ct} = \beta_0 + \beta_1 \mathbf{X}_{ct} + \beta_2 \mathbf{K}_{ct} + \beta_3 A_{ct} + \beta_4 S_{ct} + \beta_5 I_{ct} + \beta_6 \mathbf{Z}_{ct} + \beta_7 t + \beta_8 F_{ct}$$

Using this model, the authors constructed expected yield curves for each country between 1975 and 2007 (in both Mg/ha and Mkal/ha), as well as for the temperate and tropical regions, and for the world. They also constructed counterfactual yield curves for each country and for each variable, by holding that variable constant at 1975 levels. By comparing the integral of an expected yield curve with the integral of a corresponding counterfactual curve for a variable v , one can evaluate the impact of v on yield relative to other variables.

2.4.2 Decision Trees

Decision trees are machine learning tools for classification that, much like LCSs, attempt to group the outcomes of a process based on the inputs to that process. In this case, the outcomes are simply the changes in country-level crop yields and the inputs are the year-to-year changes in the values of the variables listed above. Each node in the tree corresponds to an attribute of the input (e.g. change in soil quality) and specifies a value of that attribute according to which the training examples may be partitioned into two groups: examples whose value for that attribute falls above the specified value and those whose value is less than or equal to the specified value. Beginning at the root node and partitioning in this way generates some number of leaves, each containing a subset of the examples. The objective is to construct a tree such that the examples in each leaf are as homogeneous as possible with respect to their outcomes. A path from the root node to one of the leaves thus characterizes the attribute values of the examples in that leaf. The closer to the root an attribute node is, the more it may be said to “explain” the differences in outcomes among the examples.

2.4.3 Results

Both methods found that changes in crop mix accounted for more of the growth in yield between 1975 and 2007 than any other single variable. In the tropics, an increase in the average daytime growing season temperature correlated with a noticeable decrease in yields. Perhaps surprisingly, investment in irrigation, land, and machinery and equipment, as well as the quality of the cropped soil, had a negligible impact. The methods disagreed, however, on the importance of fertilizer use: Where the econometric model showed it to have a significant positive influence on yields, the decision trees revealed no such relationship.

With all of this analysis already done, one may reasonably question the

utility of re-evaluating the same data with an LCS. There are at least two advantages:

1. LCSs have proved themselves effective in supervised learning tasks of precisely this sort and have the potential to reveal important patterns in the data not made evident by either decision tree or regression analysis.
2. An LCS may offer further support for the findings obtained through these first two methods.

3 System Design

In this section, I present my design for a Michigan-style LCS for the classification of examples with real-valued attribute vectors. The explanation consists of four parts: The first part explains the representation of rules, including a short description of their attributes, and a longer discussion about the choice of representation of the condition in particular. The second part describes the learning component and offers a general overview of the GA, as well as more detailed analysis of its constituent modules. The third part covers the testing component—the algorithm for the classification of new examples once a rule set has been created. Finally, the fourth part enumerates the parameters of the LCS.

3.1 Rule Representation

Rules are represented as objects with the following member variables:

id: A unique identifier for the rule in the population.

classification: The class, given by an integer value, that an example matching the rule will be assigned.

true_positives: The number of examples that the rule both matches and correctly classifies.

true_negatives: The number of examples that are neither matched by the rule nor of the same class.

false_positives: The number of examples that the rule matches, but incorrectly classifies.

false_negatives: The number of examples that have the same class as the rule but that the rule does not match.

num_dont_care: The number of Attribute objects in the rule’s condition (represented as a vector of Attribute objects) whose *dont_care* variable is currently set to true.

fitness_1: An initial fitness value used to determine a preliminary ranking of the rules.

fitness_2: A fitness value that is based on Fitness 1, but that takes into account redundancy (the covering of examples already covered by other, fitter rules).

condition: A vector of Attribute objects that describes the attribute values an example must have to be matched by the rule.

As mentioned in section 2.2.2, much of LCS research has concerned itself with the classification of examples with binary-valued attributes that simply indicate the presence or absence of some feature in the example. The conditions of rules used in LCSs of this sort may therefore be represented as strings of characters from the ternary alphabet $\{0, 1, \#\}$. When dealing with real values, however, determining how best to represent the condition is more difficult. With real values, not only does the search space expand to a theoretically infinite size, but the genetic operators (crossover and mutation) have to be adapted as well. Thus, the condition must be constructed in a manner that (a) reduces the search space to a reasonable size, and (b) does not hinder or over-complicate the functioning of the genetic operators.

The success of fuzzy LCSs like FCS [22] and Fuzzy-UCS [9] (a fuzzy update of the original UCS) suggests one promising approach. Given the complexity of these systems, though, and time limitations, I chose the simpler, but nonetheless well-supported method of discretizing all attributes into some number of quantiles. Each attribute in the rule’s condition not marked as irrelevant covers a range of values specified by one of the quantiles. Note that a condition constructed in this way meets both of the criteria given in the preceding paragraph: It radically restricts the search space (since each attribute will have a small number of quantiles) and it does not inhibit or fundamentally alter the behavior of the genetic operators (as will be shown in more detail further on in this section). The number of quantiles used is the same across all condition attributes and is specified by the user. The condition is represented as a vector of Attribute objects, each of which

has a name, a quantile number, an upper and lower bound describing the range of that quantile, and a boolean “don’t care” value indicating whether the attribute is relevant for the rule to which it belongs.

3.2 Rule Discovery and Learning

3.2.1 Overview

As with virtually all other LCSs, this system uses a GA in its discovery component. One complete run of the LCS attempts to characterize a single “target” class from the data. The population is initialized with a user-specified number of rules whose class attribute (action) values are set to that of the target class, and whose condition attributes are randomly either marked as irrelevant (i.e. with their *dont_care* variables set to true) or else assigned a random quantile. For each rule in the population, the system tallies the number of true positives, true negatives, false positives, and false negatives among the examples in the training set. The rule is then assigned a preliminary fitness by computing the natural log of the odds ratio and adding a small boost for each of its attributes that is marked as a “don’t care.” Once this has been done for the entire population, the rules are ranked by fitness.

Next, a second fitness is calculated using the same procedure, except that an example may count as a true positive for at most one rule. In computing the original fitness, it frequently happens that a single example will be counted as a true positive for multiple rules. In order to avoid such redundancy and to encourage diversity in the population, the second fitness counts an example as a true positive only for the fittest rule (based on the first fitness) that covers it.

The N best rules as evaluated by this second fitness are then immediately added to the next generation. To fill the remainder of the new population, $pop_size - N$ rules are selected for reproduction from the current one using stochastic universal sampling. The selected rules are paired and crossed over to produce two offspring, which are mutated and added to the next generation. The process then repeats for a user-specified number of iterations.

3.2.2 Fitness Update

Both the first and second fitnesses are computed using the natural log of the odds ratio, in conjunction with a slight bonus for each of its “don’t care” attributes. A given example is either a positive instance ($T+$) of the target class or a negative instance ($T-$). Similarly, a given rule will *classify* the

example either as a positive instance of the target class ($R+$) or a negative instance ($R-$). The four possible combinations of the example's actual class ($T+$ / $T-$) and the class that a given rule selects for that example ($R+$ / $R-$) yield the following matrix:

	$T+$	$T-$
$R+$	a	b
$R-$	c	d

Figure 1: The matrix of true positives, true negatives, false positives, and false negatives. The column indicates whether an example is a positive or negative instance of the target class. The row indicates whether the rule classifies the example as a positive or negative example.

Clockwise from the top left, the elements of the matrix correspond to true positives, false positives, true negatives, and false negatives (for a given rule and example pair). As its name suggests, the odds ratio is a ratio of two odds:

1. The odds that a rule classifies an actual positive example of the target class as a true positive (a/c).
2. The odds that a rule classifies a negative example of the target class as a true positive (b/d).

Dividing (1) by (2) gives:

$$\frac{\left(\frac{a}{c}\right)}{\left(\frac{b}{d}\right)} = \frac{ad}{bc} = \frac{|TP| \cdot |TN|}{|FP| \cdot |FN|}$$

Note that when either $|FP|$ or $|FN|$ is 0, the odds ratio is undefined. In order to avoid this problem, $|TP|$, $|TN|$, $|FP|$, and $|FN|$ are all initialized to 0.5.

There is no difference in the calculation of the two fitnesses—only in the tallying of true and false positives and negatives. In the first fitness, an example e counts as a true positive for all the rules that match it. In the second fitness, e counts as a true positive only for the *fittest* rule that matches it, as determined by the first fitness. For a rule R that matches e , but that is not the fittest that does so, e is simply not included in any of the tallies. Because the second fitness is the one that drives the genetic algorithm, rules that are redundant—that cover examples already covered

by fitter rules—will over time be weeded from the population. Observe that this schema also rewards rules that cover just a few examples, so long as those examples are not covered by fitter rules.

Finally, the odds ratio is transformed into log-space to reduce the difference between the fitnesses of the best and worst rules. If the number of true positives and negatives exceeds the number of false positives and negatives by a considerable margin, and if there are many examples in the data set, the odds ratio can become disproportionately large. Taking the log thus preserves diversity in the population by mitigating the dominance of the fittest individuals.

3.2.3 Selection

Selection is done based only on the second fitness using stochastic universal sampling (SUS), a superior alternative to traditional roulette wheel selection developed by James Baker [24]. In roulette wheel (or fitness-proportionate) selection, members of a population are selected randomly, where the probability that a member k is chosen is given by

$$\frac{\text{Fitness}(k)}{\sum_{i=1}^N \text{Fitness}(i)}$$

The basic algorithm is shown on the next page. First, the sum of the fitnesses of all members of the population is computed. Next, a number r in the range $[0, \text{fitness_sum}]$ is determined by randomly selecting a value from the interval $[0, 1]$ and multiplying that value by the sum of the fitnesses. The algorithm then iterates over the members of the population. At each iteration, the fitness of the current member is subtracted from r . The member whose fitness, when subtracted from r , causes r to fall to or below 0 is the one selected. It is clear, therefore, that the larger the fitness of an individual, the more likely it is to be chosen.

The primary shortcoming of roulette wheel selection is the extent to which it is biased against rules of lesser fitness. Generally speaking, we want the rules selected to be among the fitter ones, but in order to avoid premature convergence, some low-fitness rules must participate regularly in the crossover.

SUS reduces this bias. Instead of generating a new random value for each selection, SUS uses a single random value to make all of its selections. This turns out to be of great utility in increasing the frequency with which low-fitness rules are selected. Furthermore, it has been shown to have faster

Algorithm 1 : Roulette Wheel Selection

```
function RWS( $Pop, N$ )  
   $fitness\_sum \leftarrow \text{sumFitnesses}(Pop)$   
   $r \leftarrow \text{random}(0, fitness\_sum)$   
  for  $i$  in  $\{0, \dots, N - 1\}$  do  
     $r \leftarrow (r - \text{Fitness}(Pop[i]))$   
    if  $r \leq 0$  then  
      return  $i$   
    end if  
  end for  
  return  $N - 1$   
end function
```

Algorithm 2 : Stochastic Universal Sampling

```
function SUS( $Pop, N$ )  
   $fitness\_sum \leftarrow \text{sumFitnesses}(Pop)$   
   $pointer\_dist \leftarrow fitness\_sum / N$   
   $pointer\_start \leftarrow \text{random}(0, pointer\_dist)$   
   $pointers \leftarrow \text{newArray}(N)$   
  for  $i$  in  $\{0, \dots, N - 1\}$  do  
     $pointers[i] \leftarrow pointer\_start + (i \cdot pointer\_dist)$   
  end for  
   $selected \leftarrow \text{newArray}(N)$   
  for  $p$  in  $pointers$  do  
     $j \leftarrow 0$   
    while  $\text{sumFitnesses}(Pop[0], Pop[j]) < p$  do  
       $j \leftarrow j + 1$   
    end while  
     $selected.Add(Pop[j])$   
  end for  
end function
```

average runtime and less variance in the fitnesses of the selected rules than does roulette wheel selection [25].

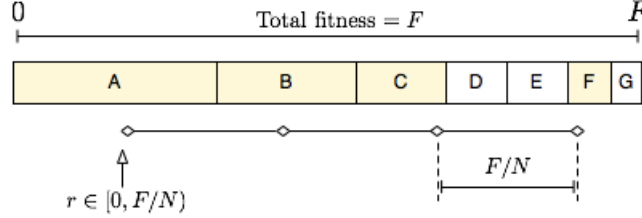


Figure 2: Beginning at a random starting point, stochastic universal sampling (SUS) uses evenly spaced pointers to select rules (Source: [26])

Algorithm 2 shows SUS in pseudocode. It will help, in understanding the behavior of SUS, to imagine the interval $[0, fitness_sum]$ on the real number line. Additionally, imagine that each member of the population is given its own subinterval within the $[0, fitness_sum]$ that is equivalent in length to its fitness (see figure 2).

To begin, a value known as the “pointer interval” is computed by dividing the sum of the fitnesses ($fitness_sum$) by the number of rules to be selected (N). Then, a random number r is chosen from the range $[0, pointer_interval]$. Beginning at r , the algorithm selects N positions along $[0, fitness_sum]$ by counting out intervals of length $pointer_interval$. The rules corresponding to the subintervals in which the positions fall are the ones selected.

3.2.4 Genetic Operators and Replacement

At each iteration, a fixed number of the fittest rules from the previous generation are added to the next generation. This number is expressed as a fraction of the population size in the *elitism_rate* parameter, and the rules thus selected are said to be “elite.” The rest of the population for the next generation is filled via crossover and mutation of the rules chosen to reproduce during the selection phase.

Crossover. The GA uses single-point crossover. For every set of parents, a random integer in the range $[1, num_attributes - 1]$ is selected as the crossover point. The condition of each of the offspring rules is created by swapping the attribute values to the right of the crossover point in each of the parents. The values of the attributes’ *dont_care* variables are also preserved.

Mutation. After crossover, the mutation operator iterates over the condition attributes of each offspring. Each attribute has the same probability of being modified (p_mutate). If it has been determined that an attribute is to be mutated, a second random number in the interval $[0, 1]$ is generated. If that number is less than or equal to a probability p_dont_care , the attribute's *dont_care* variable is set to true. If the number is greater than p_dont_care , the attribute is randomly assigned a new quantile.

After both operators have been applied, all of the offspring replace all of the non-elite ($pop_size - N$) members of the current generation. The resulting population will be the one used in the next generation.

3.3 Classification of Examples

The classification of examples in the test set is very straightforward. At the end of *num_iters* generations, the genetic algorithm terminates. Only the rules in the final population that (a) are elite and that (b) have more true positives than false positives are used for the classification task. There are several reasons for these criteria:

1. The elite rules in the final population are (mostly) those that have weathered repeated applications of the selection procedure and have proved themselves effective. Among the non-elites, which will have just been generated through crossover and mutation on the final iteration, there are likely to be many bad rules.
2. We are interested in minimizing the number of rules needed to accurately characterize the target class.
3. Rules that incorrectly identify negative instances more frequently than they correctly identify positive ones contribute would only diminish the accuracy of the system.

The number of rules used for the classification may therefore never exceed $pop_size * elitism_rate$. Once these rules have been identified, the LCS iterates over the examples in the test set. If a rule that covers the current example is found among the set selected, the example is classified as a member of the target class. If no matching rule is found, the example is classified as a member of the default class.³ After a class has been selected for the

³Since the system attempts to characterize just one class per run (the target class), it does not matter what is chosen for the default class, so long as it is different from the target class.

example, the system compares the selected class with the actual class of the example and categorizes it accordingly as a true positive, true negative, false positive, or false negative. The tallies of all four categories are outputted once all examples have been classified.

3.4 Parameters

The system takes the following arguments:

pop_size: the number of rules in the population.

num_iters: the number of iterations the LCS will run before terminating.

target_class: the class in the training data that the LCS will attempt to characterize.

elitism_rate: the approximate fraction of the population that is to be retained from one generation to the next.

p_mutate: the probability that a condition attribute will be mutated when the mutation operator is invoked.

p_dont_care: if it has been determined that an attribute is to be mutated, the probability that its *dont_care* variable will be set to true (instead of assigning it a new quantile).

training_set: a Dataset object containing all of the examples in the training set and other relevant information.

test_set: a Dataset object containing all of the examples in the test set and other relevant information.

4 Experiment and Results

4.1 Preliminary Evaluation on Small Data Sets

As a benchmark test of performance, I ran the system on the Iris and Wine data sets from the UC-Irvine Machine Learning Repository [27, 28]. Details of the data sets are shown in tables 2 and 4 in section 6.⁴ For both sets, quartiles were computed for each non-class attribute based on the range of values for that attribute across all examples in the target class. The population

⁴All tables mentioned hereafter can be found in section 6.

was then initialized with *pop_size* rules whose attributes are assigned random quartile values. Trial runs revealed that parameter values of *pop_size* = 40, *num_iters* = 1000, *elitism_rate* = 0.6, *p_mutate* = 0.25, and *p_dont_care* = 0.3 work well. Ten experiments were run per class on each set using a simple holdout method with training and test sets of approximately equal size. Tables 3 and 5 show the mean accuracy, odds ratio, true positives, true negatives, false positives, false negatives, and number of rules used for classification for Iris and Wine, respectively.

The LCS performed fairly on these simpler sets, achieving a weighted mean accuracy of 94.1% for the Iris test sets and 89.9% for the Wine sets. In both cases, the number of rules used for classification was reasonably compact (about 5.1 per class on Iris and 8.4 per class on Wine). Although a discrepancy in performance is to be expected between training and testing sets in most supervised learning tasks, the accuracy difference between those sets in both cases hinted at potential problems with overfitting. This would be somewhat surprising considering the selection pressure for more general rules (via the fitness bonus for those rules with more “don’t care” attributes). To make any confident statements on this point, however, required results from tests on the yield data.

4.2 Experiments on Yield Data

4.2.1 Setup

For the yield data, I conducted five ten-fold cross validations per data set per class with the same twelve sets as were used in Nelson and Congdon’s experiments. All twelve are derived from the same original set, but each represents a unique combination of the following variables:

1. *Year*. Indicates whether or not the year is included as an attribute of the examples. The purpose of splitting the data in this way is to evaluate the effect that time exhibits on yield changes, independent of the other variables. This variable essentially comprises all other factors that influence yield but that are not considered as attributes of the data.
2. *Region*. Indicates whether the data set includes data from countries in the tropics, countries in the temperate regions, or both (global). Looking at the data by region may reveal differences in which attributes have the most significant influence on yield.

3. *Yield Units*. Indicates whether the change in yield is expressed as Mg/ha or Mkal/ha. Framing the data in terms of both mass and energy simply offers different ways of understanding the yield.

Thus, two possibilities each for the *year* and *yield units* variables, and three possibilities for the *region* variable gives twelve data sets. The class attribute, change in yield, was discretized to tertiles (“high” (H), “medium” (M), and “low” (L) change) as it was in Nelson and Congdon’s work. The non-class attributes, however, were discretized into quartiles, after some preliminary tests showed them to be preferable to tertiles, quintiles, and sextiles.

Good parameter values were determined through experimentation and consultation of the literature. Ultimately, I maintained the values of *elitism_rate* (0.6), *p_mutate* (0.25), and *p_dont_care* (0.3) used on the Iris and Wine sets, but increased *pop_size* to 300 and *num_iters* to 2500.

4.2.2 Discussion

Tables 7 through 18 show averages of the results of the five ten-fold cross validations. A quick survey of the data reveals several interesting patterns.

First, the system exhibits a reasonably high degree of accuracy on the training sets, particularly on the temperate and tropical data, where it achieved mean accuracies of 0.950 and 0.961, respectively. Performance on the global data was noticeably worse (with a mean accuracy of about 0.873), but such a result is unsurprising, considering that the global sets included both temperate and tropical examples, which renders the task of generating accurate, yet general rules substantially more difficult. The strength of the system’s performance on the training sets is perhaps most evident in its near-complete avoidance of false positives. This is no doubt partly a consequence of the restriction that all rules to be used in the classification task must have more true positive examples than false positives ones. But even considering this criterion, one might still expect to see *some* false positive examples. I return to this topic below.

Second, performance declined substantially on the test sets. The mean accuracy was 0.664 on the temperate data, 0.594 on the tropical, and 0.666 on the global. These observations, in conjunction with the relatively strong performance on the training set, seem to reinforce the conclusion that the system struggles with overfitting — a conclusion supported by the experiments on the Wine and Iris data sets, albeit to a lesser extent.

One could potentially mitigate overfitting by allowing more rules with more false positive examples in the training set. But the requirement that

rules used for classification have a greater number of true positives than false positives appears to be a reasonable one. How to

5 Future Work and Conclusion

In this paper, I presented a Michigan-style, real-valued LCS designed to establish a relationship between the change in various country-level agricultural inputs and the change in overall agricultural yield. Specifically, I investigated data collected by Erik Nelson, previously analyzed in a 2016 paper by Nelson and Congdon. In section 2, I offered some background on the GBML paradigm, on LCSs in general, and on several milestones in the research on LCSs for supervised learning. In section 3, I outlined the design of my own LCS, and in section 4 I reported on the performance of the LCS both on sample data sets from the UCI Machine Learning Repository and on Nelson’s agricultural data. In the remainder of this final section, I discuss directions for future work.

5.1 Accuracy

The results of this project reveal many potential avenues for improvement and further experimentation. Plainly, the performance of the system on the ten-fold cross validations indicates that changes must be made to improve accuracy. Several possibilities suggest themselves.

For one, it is conceivable that the system could be enhanced by using a different fitness function. Although the log of the odds ratio makes a good deal of intuitive sense, performance can nonetheless be hindered when the best rules in the population have fitnesses that exceed those of the less effective rules, as the top few rules will tend to dominate and result in premature convergence. Rank-based fitness, in which individuals are assigned a fixed fitness value based on their rank in the population, could provide a solution to this problem. Initial tests using a naive rank-based fitness algorithm produced mixed results, but it is conceivable that more sophisticated algorithms would yield significant gains in accuracy.

The system may further benefit from additional genetic operators. An operator like Pier Lanzi’s *specify*, which periodically generates rules directly from examples, could help to counteract premature convergence and to encourage greater coverage of the training set [15].

Lastly, an alternative attribute representation could also contribute to greater accuracy. Given that the attributes of Nelson’s data are all continuous-valued, a fuzzy representation such as the one proposed in Valenzuela-

Rendón’s FCS could allow for more flexible rules, as well as a more descriptive characterization of the change in yield [22].

5.2 Data

In addition to accuracy improvements, the project would also benefit from more and different kinds of data. Nelson’s data covers the years from 1975 to 2007 and comprises many variables that clearly have a significant impact on output. But examining data that covers a greater time range and that includes metrics for other relevant factors (e.g. GDP, population, presence of plant disease) would allow for a better understanding of broader agricultural trends.

6 Figures and Tables

All entries in the tables apart from the “Type” and “Class” are averages across ten tests. “Type” denotes the kind of set (testing or training); “Class” refers to the target class; “Size” is the average number of examples in the target class; “Acc” is the overall accuracy; “lnOR” is the average of the natural log of the odds ratio; “TP,” “TN,” “FP,” and “FN,” are the average numbers of true positives, true negatives, false positives, and false negatives; and “# Rules” indicates the average number of rules actually used to classify the examples.

6.1 Iris and Wine

The results shown in tables (3) and (4) reflect averages across ten holdout experiments for each class.

Iris	
Number of Examples:	150
Classes:	Iris-Setosa (50), Iris-Versicolour (50), Iris-Virginica (50)
Number of Non-Class Attributes:	4
Non-Class Attribute Value Types:	Real
Non-Class Attributes:	sepal length, sepal width, petal length, petal width

Table 2: The Iris dataset from the UCI Machine Learning Repository.

Iris									
Type	Class	Size	Acc	lnOR	TP	TN	FP	FN	# Rules
Training	Setosa	25.0	1	8.553	25.0	50.4	0	0	4.2
	Versicolour	25.2	0.981	7.218	24	50	0.2	1.2	5.9
	Virginica	25.2	0.981	7.558	24	50.2	0	1.2	5.3
Testing	Setosa	25.0	0.983	7.570	24.5	48.8	0.8	0.5	4.2
	Versicolour	24.8	0.89	4.120	19.8	26.6	3.2	5.0	5.9
	Virginica	24.8	0.95	5.664	22.1	48.8	1	2.7	5.3

Table 3: Averages of 10 holdout tests for each class in the Iris data set.

Wine	
Number of Examples:	178
Classes:	class 1 (59), class 2 (71), class 3 (48)
Number of Non-Class Attributes:	13
Non-Class Attribute Value Types:	Integer, Real
Non-Class Attributes:	Alcohol, Malic Acid, Ash Alcalinity of Ash, Magnesium, Total phenols, Flavanoids, Nonflavanoid phenols, Proanthocyanins, Color intensity, Hue, OD280/OD315 of diluted wines, Proline

Table 4: The Wine dataset from the UCI Machine Learning Repository. The names for the class attribute values are not specified.

Wine									
Type	Class	Size	Acc	lnOR	TP	TN	FP	FN	# Rules
Training	1	30.3	1	8.895	30.3	58.8	0	0	9.7
	2	35.0	1	8.955	35.0	54.1	0	0	8.7
	3	23.8	1	8.763	23.8	65.3	0	0	6.4
Testing	1	29.7	0.841	3.333	24.8	50.0	9.2	4.9	9.7
	2	35.0	0.919	4.894	30.4	51.3	2.6	4.6	8.7
	3	24.2	0.939	5.626	20.7	62.8	1.9	3.5	6.4

Table 5: Averages of 10 holdout tests for each class in the Wine data set.

6.2 Crop Yields

6.2.1 Attributes

Table 6 shows the attributes of the yield data. All but four of the attributes (“year,” “tropical,” “Mkcal/ha,” “Mg/ha”) are present in all of the data sets. Recall that half of the data sets express change in yield in Mkcal/ha,

Attribute	Description
year*	
tropical*	An indicator denoting whether the country is tropical or temperate.
soil	The composite soil quality score (on a 1 to 5 scale, with lower numbers indicating better soil).
ha	Total cropped hectares.
rice	Cropped rice as a percentage of total cropped area.
wheat	Cropped wheat as a percentage of total cropped area.
sugar	Cropped sugar as a percentage of total cropped area.
grains	Cropped coarse grains as a percentage of total cropped area.
oil	Cropped oil crops as a percentage of total cropped area.
fruits	Cropped fruits as a percentage of total cropped area.
roots	Cropped roots and tubers as a percentage of total cropped area.
other	All other cropped produce as a percentage of total cropped area.
davg	Average daytime growing season temperature.
navg	Average nighttime growing season temperature.
pavg	Total rainfall (in mm) over cropped lands during the growing season.
irr	Fraction of cropped lands equipped for irrigation.
land	Total money (in 2005 USD) invested in agricultural land development divided by cropped hectares.
eqp	Total money (in 2005 USD) invested in agricultural equipment divided by cropped hectares.
Mkcal/ha*	Total crop yield in metric tons per hectare.
Mg/ha*	Total crop yield in millions of kilocalories per hectare.

Table 6: Attributes of the crop yield data. Attribute names followed by a “*” indicate that the attribute is not used in all data sets. Descriptions were taken with some modification from [3].

while the other half express it in Mg/ha. Also recall that half of the data sets include the year, and half do not. Only the four global data sets use

the “tropical” attribute, which simply indicates whether a given example comes from a tropical country or a temperate one. Except for “year” and “tropical,” each of the attributes in the table should be interpreted as the annual *change* in the metric described — not as an absolute quantity.

6.2.2 Results

The results for each class in tables 7 through 18 reflect averages across five ten-fold cross validations. Tables 7 through 10 show results for temperate countries, tables 11 through 14 for tropical countries, and 15 through 18 for all countries. In all experiments, the LCS used the maximum number of rules allowed given the parameters ($pop_size = 300, elitism_rate = 0.6 \rightarrow 180$ rules). Consequently, the “# Rules” column has been omitted. The number of examples in each set is shown in parentheses in the “Type” column.

temperate, with year, Mkal/ha								
Type	Class	Size	Acc	lnOR	TP	TN	FP	FN
Training (1872)	H	591.3	0.957	9.71	511.9	1280.7	0	79.4
	M	675.0	0.939	9.35	560.6	1197.0	0	114.4
	L	605.2	0.953	9.59	517.6	1266.8	0	87.6
Testing (208)	H	65.7	0.701	1.40	35.7	110.2	32.1	30.0
	M	75	0.659	1.08	36.2	101.0	32.0	38.8
	L	67.8	0.676	1.16	34.4	106.0	34.2	33.4

Table 7: Performance on examples from countries in temperate regions, with the year attribute included, and with yield measured in Mkal/ha.

temperate, no year, Mkal/ha								
Type	Class	Size	Acc	lnOR	TP	TN	FP	FN
Training (1872)	H	591.3	0.956	9.68	510.0	1280.7	0	81.3
	M	675.0	0.938	9.35	558.5	1197.0	0	116.5
	L	603.4	0.953	9.61	915.6	1268.6	0	87.8
Testing (208)	H	59.1	0.650	1.32	32.8	102.3	28.6	27.6
	M	75.0	0.658	1.08	35.98	101.2	31.8	40.0
	L	69.6	0.664	1.06	34.6	103.6	34.8	35.1

Table 8: Performance on examples from countries in temperate regions, with the year attribute *excluded*, and with yield measured in Mkal/ha.

temperate, with year, Mg/ha								
Type	Class	Size	Acc	lnOR	TP	TN	FP	FN
Training (1872)	H	529.2	0.965	9.85	463.6	1342.8	0	65.6
	M	746.1	0.939	9.42	631.2	1125.9	0	114.8
	L	597.0	0.951	9.52	504.7	1275.0	0	92.2
Testing (208)	H	58.8	0.673	1.296	30.4	115.5	33.7	28.4
	M	82.9	0.634	0.956	40.9	91.0	34.1	42.0
	L	66.0	0.702	1.122	32.9	107.1	34.9	33.2

Table 9: Performance on examples from countries in temperate regions, with the year attribute included, and with yield measured in Mg/ha.

temperate, no year, Mg/ha								
Type	Class	Size	Acc	lnOR	TP	TN	FP	FN
Training (1872)	H	529.2	0.965	9.85	463.8	1342.8	0	65.38
	M	746.1	0.935	9.36	625.1	1125.9	0	121.0
	L	596.0	0.951	9.55	504.2	1276.0	0	91.8
Testing (208)	H	53.5	0.636	1.24	28.36	103.9	28.8	25.1
	M	82.9	0.634	0.97	41.9	89.8	35.3	41.0
	L	67.0	0.679	1.20	34.6	106.5	34.5	32.4

Table 10: Performance on examples from countries in temperate regions, with the year attribute *excluded*, and with yield measured in Mg/ha.

tropics, with year, Mkal/ha								
Type	Class	Size	Acc	lnOR	TP	TN	FP	FN
Training (1670)	H	589.5	0.957	9.641	516.9	1080.9	0	72.6
	M	505.8	0.967	9.840	450.1	1164.6	0	55.7
	L	576.8	0.959	9.68	508.9	1093.5	0	67.9
Testing (186)	H	65.5	0.577	0.357	27.6	79.5	40.6	37.9
	M	56.2	0.595	0.349	22.4	81.1	41.3	33.8
	L	62.2	0.586	0.407	26.6	82.2	41.3	35.6

Table 11: Performance on examples from countries in tropical regions, with the year attribute included, and with yield measured in Mkal/ha.

tropics, no year, Mkal/ha								
Type	Class	Size	Acc	lnOR	TP	TN	FP	FN
Training (1670)	H	589.5	0.95	9.562	511.8	1080.9	0	77.7
	M	505.8	0.964	9.76	445.9	1164.6	0	59.9
	L	575.6	0.958	9.66	505.29	1094.8	0	70.3
Testing (186)	H	65.5	0.569	0.256	25.8	79.9	40.2	39.7
	M	56.2	0.595	0.317	21.7	88.8	40.6	34.5
	L	63.4	0.590	0.449	27.4	82.1	40.1	36.0

Table 12: Performance on examples from countries in tropical regions, with the year attribute *excluded*, and with yield measured in Mkal/ha.

tropics, with year, Mg/ha								
Type	Class	Size	Acc	lnOR	TP	TN	FP	FN
Training (1670)	H	651.6	0.953	9.60	572.4	1018.8	0	79.2
	M	434.7	0.976	10.09	393.8	1235.7	0	40.9
	L	583.4	0.960	9.72	516.3	1086.8	0	67.2
Testing (186)	H	72.4	0.577	0.45	32.4	74.7	38.5	40.0
	M	48.3	0.643	0.591	19.3	100.1	37.2	29.0
	L	65.6	0.599	0.55	29.2	82.0	38.2	36.3

Table 13: Performance on examples from countries in tropical regions, with the year attribute included, and with yield measured in Mg/ha.

tropics, no year, Mg/ha								
Type	Class	Size	Acc	lnOR	TP	TN	FP	FN
Training (1670)	H	651.6	0.952	9.58	571.4	1018.8	0	80.2
	M	434.7	0.973	9.94	390.2	1235.7	0	44.5
	L	584.4	0.96	9.71	516.7	1086.1	0	67.7
Testing (186)	H	72.4	0.572	0.415	31.7	74.5	38.7	40.7
	M	48.3	0.639	0.517	18.4	100.3	37.0	29.9
	L	64.6	0.587	0.429	27.9	81.0	39.9	36.7

Table 14: Performance on examples from countries in tropical regions, with the year attribute *excluded*, and with yield measured in Mg/ha.

global, with year, Mkcal/ha								
Type	Class	Size	Acc	lnOR	TP	TN	FP	FN
Training (3542)	H	1180.8	0.873	8.947	731.2	2361.6	0	449.6
	M	1180.8	0.865	8.849	703.9	2361.6	0	476.9
	L	1182.9	0.872	8.94	730.2	2359.5	0	452.7
Testing (394)	H	131.2	0.670	0.96	44.4	219.2	43.2	87.8
	M	131.2	0.655	0.76	40.16	217.6	44.8	91.0
	L	129.1	0.670	0.92	42.7	220.9	43.6	86.4

Table 15: Performance on examples from all countries, with the year attribute included, and with yield expressed in Mkcal/ha.

global, no year, Mkcal/ha								
Type	Class	Size	Acc	lnOR	TP	TN	FP	FN
Training (3542)	H	1180.8	0.874	9.00	733.6	2361.6	0	447.2
	M	1180.8	0.866	8.829	704.4	2361.6	0	476.4
	L	1178.6	0.871	8.86	723.2	2363.7	0.1	455.4
Testing (394)	H	131.2	0.667	0.93	44.1	218.4	44.0	87.1
	M	131.2	0.659	0.80	40.0	219.5	42.9	91.4
	L	133.4	0.662	0.90	42.6	218.0	42.2	90.8

Table 16: Performance on examples from all countries, with the year attribute *excluded*, and with yield expressed in Mkcal/ha.

global, with year, Mg/ha								
Type	Class	Size	Acc	lnOR	TP	TN	FP	FN
Training (3542)	H	1180.8	0.880	9.04	756.3	2361.6	0	424.5
	M	1180.8	0.874	9.00	732.7	2361.6	0	448.1
	L	1179.3	0.873	8.94	729.3	2363.1	0	450.0
Testing (394)	H	131.2	0.676	1.03	45.7	220.3	42.1	85.5
	M	131.2	0.661	0.85	41.9	218.2	44.2	89.3
	L	132.7	0.662	0.88	43.6	216.8	44.1	89.0

Table 17: Performance on examples from all countries, with the year attribute included, and with yield expressed in Mg/ha.

global, no year, Mg/ha								
Type	Class	Size	Acc	lnOR	TP	TN	FP	FN
Training (3542)	H	1180.8	0.880	9.01	754.5	2361.6	0	426.3
	M	1180.8	0.872	8.93	727.2	2361.6	0	453.6
	L	1182.1	0.871	8.92	724.9	2360.4	0	457.2
Testing (394)	H	131.2	0.672	0.99	45.3	219.2	43.4	85.9
	M	131.2	0.660	0.84	42.6	217.3	45.1	88.6
	L	129.9	0.674	0.98	43.4	221.7	41.9	86.5

Table 18: Performance on examples from all countries, with the year attribute *excluded*, and with yield expressed in Mkcal/ha.

References

- [1] EPA, “Climate Impacts on Agriculture and Food Supply,” 2017.
- [2] U. N. D. of Economic and S. Affairs, “World population projected to reach 9.7 billion by 2050,” July 2015.
- [3] E. Nelson and C. B. Congdon, “Measuring the Relative Importance of Different Agricultural Inputs to Global and Regional Crop Yield Growth Since 1975,” *Economics Department Working Paper Series*, Sept. 2016.
- [4] T. Kovacs, “Genetics-Based Machine Learning,” in *Handbook of Natural Computing* (G. Rozenberg, T. Bck, and J. N. Kok, eds.), pp. 937–986, Springer Berlin Heidelberg, 2012. DOI: 10.1007/978-3-540-92910-9_30.
- [5] J. H. Holland, *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*, vol. viii. Oxford, England: U Michigan Press, 1975.
- [6] J. H. Holmes, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, “Learning classifier systems: New models, successful applications,” *Information Processing Letters*, vol. 82, pp. 23–30, Apr. 2002.
- [7] J. H. Holland, “Properties of the Bucket Brigade,” in *Proceedings of the 1st International Conference on Genetic Algorithms*, (Hillsdale, NJ, USA), pp. 1–7, L. Erlbaum Associates Inc., 1985.
- [8] C. J. C. H. Watkins, *Learning from Delayed Rewards*. PhD thesis, 1989.
- [9] A. Orriols-Puig, J. Casillas, and E. Bernado-Mansilla, “Fuzzy-UCS: A Michigan-Style Learning Fuzzy-Classifer System for Supervised Learning,” *IEEE Transactions on Evolutionary Computation*, vol. 13, pp. 260–283, Apr. 2009.
- [10] X. Llor, R. Reddy, B. Matesic, and R. Bhargava, “Towards Better Than Human Capability in Diagnosing Prostate Cancer Using Infrared Spectroscopic Imaging,” in *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, GECCO ’07, (New York, NY, USA), pp. 2098–2105, ACM, 2007.
- [11] Y. Gao, J. Z. Huang, H. Rong, and D. Gu, “Learning Classifier System Ensemble for Data Mining,” in *Proceedings of the 7th Annual Workshop*

on *Genetic and Evolutionary Computation*, GECCO '05, (New York, NY, USA), pp. 63–66, ACM, 2005.

- [12] Hai H. Dam, Hussien A. Abbass, and Chris Lokan, “BCS: Bayesian Learning Classifier System,” Tech. Rep. TR-ALAR-200604005, The Artificial and Adaptive Robotics Laboratory, School of Information Technology and Electrical Engineering, University of New South Wales, Cardiff, UK, 2006.
- [13] S. W. Wilson, “Classifier Fitness Based on Accuracy,” *Evolutionary Computation*, vol. 3, pp. 149–175, June 1995.
- [14] E. Bernad-Mansilla and J. M. Garrell-Guiu, “Accuracy-based learning classifier systems: models, analysis and applications to classification tasks,” *Evolutionary Computation*, vol. 11, no. 3, pp. 209–238, 2003.
- [15] P. L. Lanzi, “A Study of the Generalization Capabilities of XCS,” in *In Proceedings of the Seventh International Conference on Genetic Algorithms*, pp. 418–425, Morgan Kaufmann, 1997.
- [16] M. Keijzer, J. J. Merelo, G. Romero, and M. Schoenauer, “Evolving Objects: A General Purpose Evolutionary Computation Library,” in *Artificial Evolution*, pp. 231–242, Springer Berlin Heidelberg, Oct. 2001. DOI: 10.1007/3-540-46033-0_19.
- [17] R. J. Urbanowicz and J. H. Moore, “Learning Classifier Systems: A Complete Introduction, Review, and Roadmap,” *J. Artif. Evol. App.*, vol. 2009, pp. 1:1–1:25, Jan. 2009.
- [18] P. L. Lanzi and R. L. Riolo, “A Roadmap to the Last Decade of Learning Classifier System Research (From 1989 to 1999),” in *Learning Classifier Systems* (P. L. Lanzi, W. Stolzmann, and S. W. Wilson, eds.), no. 1813 in Lecture Notes in Computer Science, pp. 33–61, Springer Berlin Heidelberg, 2000. DOI: 10.1007/3-540-45027-0_2.
- [19] S. W. Wilson and D. E. Goldberg, “A Critical Review of Classifier Systems,” in *Proceedings of the Third International Conference on Genetic Algorithms*, (San Francisco, CA, USA), pp. 244–255, Morgan Kaufmann Publishers Inc., 1989.
- [20] S. W. Wilson, “State of XCS Classifier System Research,” in *Learning Classifier Systems* (P. L. Lanzi, W. Stolzmann, and S. W. Wilson, eds.), no. 1813 in Lecture Notes in Computer Science, pp. 63–81, Springer Berlin Heidelberg, 2000. DOI: 10.1007/3-540-45027-0_3.

- [21] J. D. Schaffer and J. J. Grefenstette, “Multi-objective Learning via Genetic Algorithms,” in *Proceedings of the 9th International Joint Conference on Artificial Intelligence - Volume 1, IJCAI’85*, (San Francisco, CA, USA), pp. 593–595, Morgan Kaufmann Publishers Inc., 1985.
- [22] Manuel Valenzuela-Rendn, “The Fuzzy Classifier System: A Classifier System for Continuously Varying Variables,” (San Diego, CA), 1991.
- [23] J. H. Holland and J. S. Reitman, “Cognitive Systems Based on Adaptive Algorithms,” *SIGART Bull.*, pp. 49–49, June 1977.
- [24] B. J., “Reducing Bias and Inefficiency in the Selection Algorithm,” in *Proceedings of the Second International Conference on Genetic Algorithms and their Application*, (Hillsdale, New Jersey), pp. 14–21, L. Erlbaum Associates, 1987.
- [25] Tobias Blickle and Lothar Thiele, “A Comparison of Selection Schemes used in Genetic Algorithms,” Tech. Rep. 11, Computer Engineering and Communication Networks Lab, Swiss Federal Institute of Technology (ETH), Zurich, Dec. 1995.
- [26] S. Hatthou, “A diagram illustrating the Stochastic Universal Sampling operation,” Jan. 2007.
- [27] “UCI Machine Learning Repository: Iris Data Set.”
- [28] “UCI Machine Learning Repository: Wine Data Set.”