Evaluating the Impact of Various Agricultural Inputs on Crop Yields Using Learning Classifier Systems

Will Gantt

Clare Bates Congdon

11/30/16

***Project Description***

The goal of my project is to use learning classifier systems (LCSs) to measure the impact of different variables on regional and global crop yields. LCSs are systems that use evolutionary computation to evolve a set of rules for classifying inputs based on a specified attribute. In an effective LCS, these rules will indicate what features of the inputs are most relevant in determining how they should be classified. By constructing an LCS for yield data from 1975 to 2007 (collected by Erik Nelson), I will identify the environmental and economic factors that most significantly influence changes in crop yields. This project aims to expand on the analysis conducted in [1].

**I. Evolutionary Computation**

Evolutionary computation (EC) is an artificial intelligence problem-solving approach that comprises a group of algorithms (often called *evolutionary algorithms*) modeled on biological systems. Of these, genetic algorithms (GAs) are among the most enduring. Originally developed by John Holland in 1975, GAs draw inspiration from Darwinian evolution and identify solutions to a problem by evolving a population of candidate solutions over a number of generations [2]. In the canonical GA, these solutions are represented as bit strings, though they may take other forms. By repeatedly subjecting candidates to processes that simulate genetic mutation, crossover, selection, and reproduction, GAs ideally generate populations of solutions that are increasingly “fit” with respect to the selection criteria. An outline of a simple GA is shown below.

1. *Initialization*. Generate a population of random candidate solutions (*individuals*).
2. *Evaluation*. Evaluate the fitness of each individual according to some fitness function *f*.
3. *Selection*. Select *n* individuals from the population to reproduce. An individual’s likelihood of being selected is proportional to its fitness.
4. *Crossover*. Randomly pair up the *n* selected individuals. Combine the bits from one member of the pair with those of the other (via one-point crossover) to produce two new solutions (or *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. target fitness, number of generations, runtime) is met.

GAs have a number of desirable characteristics. For real-world problems, computing an optimal solution directly is often prohibitively expensive. This is particularly true of many economic and statistical models that attempt to consider the impacts of a large number of variables. In these scenarios, GAs can provide a comparatively fast and inexpensive approximation of an optimum. Moreover, real-world problems routinely contain noisy data. The robustness of GAs allows them to handle such data more adroitly than other approaches; outliers and minor adjustments to inputs are unlikely to yield dramatic changes in output, since GAs evolve a *population* of solutions that changes only gradually. Finally, GAs are general enough to find application in a wide array of fields including bioinformatics, economics, physics, and climatology [3].

**II. Machine Learning**

Although this project applies a very specific problem-solving framework to a very specific problem, it is worth placing both in the broader methodological context of machine learning.

At its core, machine learning attempts to teach computers to teach themselves. There are many ways to categorize approaches to learning in computer science, but one of the most common and most useful distinctions centers on how much feedback is required of the user or programmer. This question divides approaches into three primary taxa: supervised learning, reinforcement learning, and unsupervised learning.

*Supervised learning* (SL) refers to systems in which the desired output is known for a given set of inputs. Here, the learner is rewarded or punished in proportion to how closely its outputs resemble the ones desired. Many classification problems are of this type, including spam detection, and speech and handwriting recognition [4].

For other problems, we may instead be interested in optimizing the performance of an agent on a particular task. The term “agent” is here used in its broadest sense, and may refer to a variety of learning systems, such as chess players or consumers in an economic model. An agent interacts with its environment by taking actions that move it between different environment states, where every state transition has some reward or penalty associated with it. These are typically determined by the programmer and reflect how much closer a given transition brings the agent toward achieving its goal. Ideally, the rewards and penalties are in line with the attainment of the goal, and thus the agent, by seeking to maximize its net reward, will learn the optimal series of actions. This approach, known as *reinforcement learning* (RL), adopts a fundamentally different philosophy from that of SL and tends to address a different set of problems. Where SL uses a large set of examples to discern an underlying set of rules that characterize different labels, RL emphasizes more hands-off, experiential learning in which the learner is guided only by its goal of maximizing reward.

Labeling data almost always requires some amount of human interpretation of *un*labeled data, which is, unsurprisingly, far more abundant in the real world. Furthermore, when analyzing unlabeled data, we may not have any specific objective other than to look for interesting patterns. In the absence of both labels and goals, *unsupervised learning* (UL) can offer help. The spectrum of patterns that may be considered “interesting” and the diversity of methods used to find those patterns mean that UL encompasses a vast problem domain. The majority of UL algorithms focus on identifying statistical regularities, and employ techniques like density estimation, clustering, and neural networks to that end.

Since the task of my project is one of classification, and since the data I use are labeled, I have chosen to take an SL approach –– one based in evolutionary computation. This approach, known as *genetics-based machine learning*, is described in detail below.

**III.** **Genetics-Based Machine Learning**

***GBML and Learning Classifier Systems***

Genetics-based machine learning (GBML) applies evolutionary algorithms to major problems in machine learning. While the set of problems to which GBML has been applied is diverse, classification and function optimization are the most common varieties. Because my project deals with a classification task, classification is the only problem type that I will describe in detail. However, for more information on other applications, the reader is encouraged to consult [5], [6], and [7].

Classification is the problem of assigning labels to inputs [8]. A classifier system must take a set of inputs (typically, a vector of values for a set of attributes) and from them determine a representation of the different classes to which they belong. Insofar as we want to maximize the accuracy and simplicity of the system, we may conceive of this as an optimization problem – a task to which GAs are particularly well-suited. And indeed, they have long been a fundamental component of learning classifier systems (LCSs).

LCSs are rule-based learning algorithms; that is, they develop a list of actions to take when certain conditions are met. They use GAs to discover the rules and a learning paradigm (SL or RL) to improve them. The basic principle consists in evolving a population of rules or *classifiers* that collectively categorize an entire set of inputs. Classifiers are nearly always formatted as conditional (IF-THEN) statements and take an action (specified by the consequent) only when the antecedent is satisfied. If the goal of a particular LCS was to classify balls, for example, the system might generate the following rule:

IF (BROWN) AND (OBLONG) THEN (FOOTBALL)

Finally, as members of a population in a GA, each classifier also has a fitness value that reflects its ability to classify inputs accurately.

John Holmes and his colleagues [9] have identified four features common to virtually all LCSs:

* *Classifier population*: the rules (described above) that dictate how inputs are classified
* *Performance component*: a set of parameters that governs the way in which classifiers interact with their environment, including how fitness is to be evaluated and how actions are to be selected.
* *Credit Assignment*: a procedure for allocating reward to the best-performing classifiers in the population.
* *Discovery component*: a set of methods for discovering new rules. Rule discovery may be accomplished by numerous methods, but is always one of the primary functions of the genetic algorithm, which generates new rules through certain operators, such as crossover and mutation.

***Minimal Classifier System***

These four features give a general sense of the key mechanisms at work in an LCS, but in order to obtain a fine-grain understanding, it will help to consider an example. Though not designed for practical problem-solving purposes, the Minimal Classifier System (MCS) developed by Larry Bull in [10] provides a good theoretical model for this purpose.

At the start of the MCS algorithm, a random population of classifiers is initialized, each classifier consisting of a condition, which determines the inputs to which it applies; an action, which indicates how those inputs are to be classified; and a fitness value, which serves as a metric of how effective the classifier is. The condition is represented as a string from the ternary alphabet {0,1,#}, the action as a binary string, and the fitness as a non-negative integer.

At each iteration, the system evaluates a single input (a four-digit binary number) from the environment. Upon receiving the input, the system scans the population for classifiers whose condition matches the input value[[1]](#footnote-1); together, these form the *match set* [M]. If there is no such classifier, a mechanism known as a *covering operator* generates one with a random action. This new classifier replaces an existing one, chosen by roulette wheel selection, where the likelihood that a given classifier will be replaced is inversely proportional to its fitness.

The set [M] thus consists of subsets of classifiers with the same action. The sum of fitness values of all the members of a given subset is the *prediction value* of that subset. After [M] has been determined, the algorithm selects an action by alternating between the following two methods:

1. Choosing the action of the subset of [M] with the highest prediction value (known as *exploitation*).
2. Choosing an action randomly from [M] (known as *exploration*). On each iteration of this method, there is some probability that a GA will be activated. The GA creates new rules by using roulette wheel selection to choose pairs of parent classifiers, which produce offspring through probabilistic crossover and mutation. Each of the two offspring inherits the fitness value of one of the parents, unless mutation occurs, in which case both offspring are assigned the average of the fitnesses of their parents.

The subset of classifiers in [M] recommending the chosen action is known as the *action set* [A]. Once the action is chosen, the fitness of all members of [A] is updated. The algorithm repeats in this way for some user-specified number of generations.

***LCS Paradigms***

One of the great virtues of LCSs is their adaptability: Each of the four components offers substantial room for customization, and there are, consequently, a host of different LCS paradigms. Thorough surveys of these paradigms can be found in [11] and [12], but I will discuss just two of the most important ones here.

The “Michigan” and “Pitt” styles of LCS embody two different philosophies of population construction. Michigan systems implement the population as a single set of rules that together constitute a solution to the classification task. Every rule has a fitness (as in MCS) and rules compete with one another for a position in the final set. In Pitt systems, by contrast, the population comprises *many* rule sets, and competition takes place at this level, rather than at the level of the individual rule.

These features of Pitt systems have at least two important implications. First, since credit is assigned to whole sets of rules, bad rules may persist through many iterations because they belong to a candidate set that is otherwise highly fit. Second, the coarse-grained credit assignment scheme, in conjunction with the greater amount of computation required to evolve a single individual, means that Pitt-style LCSs are slower than their Michigan counterparts.

However, the Michigan approach has drawbacks of its own. Specifically, the classifier population will generally not converge as nicely as in Pitt, since individual rules are being perpetually mutated and replaced. This also means that a final solution in a Michigan system is likely to contain more bad rules than a final solution in a Pitt system.

***Challenges of LCSs***

Many of the difficulties one encounters in constructing an LCS are ones commonly found in other corners of machine learning. Building any kind of model from data requires fitting the model to the data in some way, and there are well-studied dangers inherent in this process. When the model attempts to account for certain idiosyncrasies of the data it was trained on –– rare events or outliers –– it *overfits* those data. In this case, even though the model may be a superb predictor of its training set, it will likely have a much harder time on new sets than a model that does not make the same attempt at a perfect fit.

The converse problem, *underfitting*, occurs when the model fails to reflect a feature of its training data that is in fact a common, broader trend among other datasets. While the ML literature reveals that overfitting is overwhelmingly the more common trouble, both are obstacles one must negotiate in conceiving an LCS.

A related but distinct issue in learning systems is *inductive bias*. Often referred to as *rule induction*, the process of discovering new rules presents an especially pernicious problem for the designer of an LCS. Any system that evaluates anything at all is, of course, biased in favor of those things that perform best according to the metric of evaluation. In other words, there must be bias of some kind if rule induction is to be possible. The difficulty consists in determining whether the bias is a desirable one. A preference for rules that depend on fewer attributes may seem sensible, for instance, but it may be the case that the best rules do not necessarily have this feature.

Bias present in the rule-discovery component of a learning algorithm is known as *preference* or *search bias*, and is contrasted with *representation bias*, which concerns the way in which rules are encoded. The expressive limits of a language necessarily restrict what knowledge a learning algorithm can acquire. Rules are frequently encoded as statements of basic propositional logic, for example, and thus an algorithm using such a representation will be unable to generate rules based in first-order logic [3].

***Advantages and Disadvantages of GBML***

As GBML is based in evolutionary computation, it shares many of the advantages and disadvantages of the latter. Like GAs, GBML performs well on noisy data and difficult search spaces, may be easily parallelized, and presents a sufficiently general problem-solving framework to be useful in diverse settings. Moreover, it exhibits a level of accuracy in classification tasks that is competitive with traditional (non-evolutionary) approaches.

Chief among the disadvantages of GBML is efficiency, since GAs tend to have significant time-complexity and consequently slow runtimes. The ease of parallelization can mitigate the problem, but a GBML approach may not be the best option for highly time-sensitive tasks.

**IV. Background and Project Description**

***Project Description***

The goal of my project is to identify the economic and environmental factors that most significantly impact changes in crop yields on both local and global scales. As the world population grows and as climate change intensifies, it is increasingly vital to the future of food safety and market stability that we have a sophisticated understanding of the variables at play in the realm of agriculture. My project will use an LCS to examine agricultural input and yield data and to predict changes in yield based on the levels of the various inputs.

***Background***

The data I will be using, compiled by Erik Nelson and his students, span the years from 1975 to 2007, and relate a number of inputs to changes in crop yields.[[2]](#footnote-2) Earlier this year, Nelson and Congdon undertook a project with the same objective as my own, but with a different methodology [1]. In their paper, they analyze the impact of individual inputs on yield via two distinct methods: fixed-effects econometric modeling and decision tree algorithms. In the first method, the authors calculate estimated yield curves for individual countries, for regions (temperate or tropical), and for the world using the method of least squares. These curves express yield (in Mg/Ha and Mkcal/Ha) as a linear combination of the levels of inputs. A separate, “counterfactual” curve is then calculated for each of the inputs, in which that input is held constant at its 1975 level, while other inputs are allowed to vary with time. By integrating over the difference in predicted yields of the two curves, one can determine the impact of a given variable on expected yield.

In the second method, the authors use decision tree algorithms. A decision tree divides outcomes into groups based on the values of the attributes that best explain those outcomes. In this case, a decision tree shows what levels of which inputs best explain a given amount of *change* in overall yield (discretized to “high” (H), “medium” (M), and “Low” (L) levels of change). At each node in the tree, the algorithm partitions outcomes based on a single value of a single attribute (e.g. >10% increase in sugar production and ≤ 10% increase in sugar production).[[3]](#footnote-3) In this way, the path leading from a leaf node (which contains a group of yield changes or “outcomes”) up to the root of the tree describes a set of attribute values that best predict those outcomes.

Both methods found that the most important factors contributing to global yield growth were crop mix (in the tropics), daytime growing season temperature, and time.[[4]](#footnote-4) The econometric models also showed fertilizer use to be a significant contributor, though the decision trees did not. Lastly, both indicated that investment in irrigation, land, and machinery had a negligible impact on growth.

***Project Details***

Congdon and Nelson thoroughly catalogue the limitations of their analysis. A substantial portion of these have less to do with the methods employed than with the data used. The authors note, for example, that their data do not allow them to take into account farmers’ reaction to climate change, and that the data extend only as far as 2007. Furthermore, there is no information available for any of the Soviet Union or Warsaw Pact countries. Congdon and Nelson observe that omissions such as these may bias their results.

Clearly, these are data-specific problems that cannot be remedied by applying a different methodology. Yet, a GBML approach would address certain other limitations. For one, GBML allows us to test the accuracy of our models much more easily than econometric techniques. With the latter, one commits to a particular method of statistical evaluation and applies the method to all of the available data. With GBML, by contrast, one partitions the data, using some portion of it to build the model and the remainder to test it. In this way, one can obtain a better sense of the effectiveness of an LCS than of a logistic regression.

It is not yet clear in what ways an LCS may prove superior to a decision tree, but an it will, at the very least, offer another perspective on the data, which will hopefully resolve the discrepancies between the results of the econometric and decision tree analyses and reinforce theri commonalities.

To implement an entire LCS from scratch would be to reinvent the wheel –– and likely quite poorly. Although the fundamental design and the significant majority of the code will be my own, I will use tools from the Evolving Objects library (EOlib) as the basis for the genetic algorithm [14]. While traditional implementations of genetic algorithms restrict the kind of data structure being evolved to a bit string, EOlib offers almost unlimited choice in this regard. Were speed a primary concern, the classifiers could be implemented as bit strings, but this would be markedly more tedious than using objects or another more sophisticated data structure. With this exception for the genetic algorithm, the remainder of the code will be entirely my own.

References

[1] E. Nelson and C. B. Congdon, “Measuring the Relative Importance of Different Agricultural Inputs to Global and Regional Crop Yield Growth Since 1975,” *Econ. Dep. Work. Pap. Ser.*, Sep. 2016.

[2] 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.

[3] “List of genetic algorithm applications,” *Wikipedia*. 10-Nov-2016.

[4] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*. Cambridge, MA: The MIT Press, 2012.

[5] T. Kovacs, “Genetics-Based Machine Learning,” in *Handbook of Natural Computing*, G. Rozenberg, T. Bäck, and J. N. Kok, Eds. Springer Berlin Heidelberg, 2012, pp. 937–986.

[6] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, 2001.

[7] A. A. Freitas, *Data Mining and Knowledge Discovery with Evolutionary Algorithms*, 2002 edition. Berlin ; New York: Springer, 2002.

[8] D. Lawrence, “Genetics-Based Machine Learning: Background and Introductory Survey of Major Paradigms.”

[9] J. H. Holmes, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, “Learning classifier systems: New models, successful applications,” *Inf. Process. Lett.*, vol. 82, no. 1, pp. 23–30, Apr. 2002.

[10] L. Bull, “Two Simple Learning Classifier Systems,” in *Foundations of Learning Classifier Systems*, L. Bull and T. Kovacs, Eds. Springer Berlin Heidelberg, 2005, pp. 63–89.

[11] R. J. Urbanowicz and J. H. Moore, “Learning Classifier Systems: A Complete Introduction, Review, and Roadmap,” *J Artif Evol App*, vol. 2009, p. 1:1–1:25, Jan. 2009.

[12] A. Fernandez, S. Garcia, J. Luengo, E. Bernado-Mansilla, and F. Herrera, “Genetics-Based Machine Learning for Rule Induction: State of the Art, Taxonomy, and Comparative Study,” *IEEE Trans. Evol. Comput.*, vol. 14, no. 6, pp. 913–941, Dec. 2010.

[13] J. R. Quinlan, “Induction of Decision Trees,” *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, Mar. 1986.

[14] M. Keijzer, J. J. Merelo, G. Romero, and M. Schoenauer, “Evolving Objects: A General Purpose Evolutionary Computation Library,” in *Artificial Evolution*, Springer Berlin Heidelberg, 2001, pp. 231–242.

1. 0’s and 1’s in the condition must match the input exactly, and a ‘#’ matches with both values. [↑](#footnote-ref-1)
2. For a full list of the inputs considered, see [1]. [↑](#footnote-ref-2)
3. The mathematics behind how a decision tree chooses an attribute to partition on are interesting, but will not be explained here. Those interested in learning more may consult [13]. [↑](#footnote-ref-3)
4. The time variable in Nelson and Congdon’s data accounts, at a coarse level, for several other variables that are not explicitly represented. Among these are advancements in agricultural technology, improved farm management practices, increased use of pesticides, globalization, and market liberalization [1]. [↑](#footnote-ref-4)