Expert Systems With Applications 218 (2023) 119529

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Genetic algorithms with self-adaptation for predictive classification of Medicare standardized payments for physical therapists

Reamonn Norat [a], Annie S. Wu [b,*], Xinliang Liu [c]

[a] Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL 32816, USA
[b] Department of Computer Science, University of Central Florida, Orlando, FL 32816, USA
[c] Department of Community and Population Health, Lehigh University, Bethlehem, PA 18015, USA

ARTICLE INFO

Keywords:
Genetic algorithm
Evolutionary computation
Self adaptation
Predictive modeling
Health care analytics
Medicare

ABSTRACT

We present a self-adaptive genetic algorithm for the problem of predicting if a Medicare standardized payment to a physical therapist will be above or below the national median. The percentage of Americans 65 and over is expected to increase in the coming years, increasing the need for physical therapy services. As a result, accurate prediction of expected Medicare payments based on local factors will be of increasing importance. A self-adaptive genetic algorithm is an evolutionary algorithm in which some or all of the algorithm's parameters are evolved over the course of its execution. Self-adaptation is a useful tool both for improving the performance of evolutionary algorithms, as well as improving usability through lessening the amount of parameter tuning required of the algorithm's user. While other self-adaptive approaches tend to focus on self-adaptation of only a few parameters, our approach self-adapts all of the parameters related to crossover and mutation. We compare the performance of our self-adaptive genetic algorithm with that of logistic regression and a canonical genetic algorithm on the problem of predicting Medicare payments. Logistic regression is a commonly used benchmark for this type of problem and a canonical genetic algorithm is included to allow us to see if any performance costs arise from the self-adaptive mechanisms. Results show that our self-adaptive genetic algorithm is effective at the classification of Medicare standardized payments to physical therapists, achieving accuracies of over 93%. Performance remains strong with training sets as small as 5% of the full data set. The problem representation used by our method allows for the identification of the relevant features for classification which means that our approach is capable of simultaneously performing classification and feature selection.

1. Introduction

In this paper, we present a new self-adaptive genetic algorithm (SAGA) and apply it to the problem of predicting when the annual standardized Medicare payment of a physical therapist (PT) will be above or below the national median. This problem is a predictive classification problem in which we seek to learn a model that can accurately map the input variables of a data point to a binary outcome. Genetic algorithms (GA) (Holland, 1975) are a sub-type of evolutionary algorithms (EA) that have been successfully applied to a variety of predictive classification problems. EAs have multiple parameters that must be effectively tuned in order to ensure satisfactory results, with GAs being no exception. In order to make the GA a more accessible analysis tool for domain specialists, we develop a GA that self-adapts all of its operator-related parameter settings, reducing the need for domain specialists to understand and tune those parameters. We investigate the effectiveness of SAGA on a predictive classification problem, comparing SAGA performance to that of a manually tuned canonical GA to examine the impact of the self-adaptation on GA performance and to logistic regression (LR) (Hosmer Jr & Lemeshow, 1989) which is a traditional analytical method for such problems in the field of health informatics.

EAs have been successfully applied to predictive classification problems for healthcare applications (Fidelis et al., 2000; Wu et al., 2019) as well as other applications such as multi-criteria inventory classification (Guvenir & Erel, 1998) and prediction of a country's natural gas usage (Kovačič & Dolenc, 2016). In addition to these real-world examples, previous work shows effective results when testing EA classification on various benchmarking data sets (Dehuri et al., 2008; Fernández et al., 2010). GAs are also used in conjunction with other ML algorithms to identify the relevant features of a classification problem. Typically, these hybrid approaches use a GA to select the relevant features of a problem then other algorithms, such as support vector

* Corresponding author.
E-mail addresses: reanorat@gmail.com (R. Norat), aswu@cs.ucf.edu (A.S. Wu), xilc21@lehigh.edu (X. Liu).

https://doi.org/10.1016/j.eswa.2023.119529
Received 31 May 2021; Received in revised form 23 December 2022; Accepted 5 January 2023
Available online 10 January 2023

machines (Min et al., 2006), random forest classifiers (Paul et al., 2017), neural networks (Jefferson et al., 2000; Shrivastava et al., 2017), linear discriminant analysis (Höglund, 2017), or LR (Vandewater et al., 2015; Zhang et al., 2018), perform the classification using the features selected by the GA.

One of the challenges to successfully using an EA is appropriately setting the parameter values for the algorithm. Optimal parameter settings vary from problem to problem, making effective parameter tuning an essential element of using an EA. An EA consists of multiple interdependent parameters that cannot be tuned independently due to potential non-linear interactions among the parameters (Grefenstette, 1986; Schaffer et al., 1989). Because of this interdependency, a combinatorially exhaustive search is necessary to fully optimize the parameter settings. This combinatorial search is potentially hampered by the discrete values that the user chooses for the search, which may result in missing the optimal settings.

In order to reduce the need for manual parameter tuning and to improve EA usability for non-practitioners, we investigate the effectiveness of allowing a EA to automatically adjust its own parameter settings during a run. This self-adaptive approach means that the EA evolves not only candidate solutions to a target problem but also many of the parameter values that define how the algorithm operates. The idea of self-adaptation within the field of EAs is not new (Bäck, 1992a, 1992b; Contreras-Bolton & Parada, 2015; Hinterding, 1997; Kivijärvi et al., 2003; Murata & Ishibuchi, 1996; Serpell & Smith, 2010; Smith & Fogarty, 1996; Spears, 1995; Yoon & Moon, 2002); however, these previous works each apply self-adaptation to a limited subset of the parameters. Our approach expands upon previous work by self-adapting all parameters relating to genetic operators and operator rates.

We investigate the effectiveness of SAGA on predictive classification problems and show that SAGA is able to adapt parameter values to produce competitive performance as compared to LR and to a canonical GA. We compare with LR because it is a commonly used benchmark for classification problems that map multiple independent variables to a binary outcome (Chaurasia & Pal, 2014; De Vasconcelos et al., 2001; Min et al., 2006; Serban et al., 2011; Thornblade et al., 2018). We compare with a canonical GA to verify that SAGA can perform as well as a manually tuned GA without the need to tune as many parameters as a GA. Our primary measurement is classification accuracy but, beyond these basic results, we are also interested in analyzing the behavior of SAGA on this problem. Specifically, we analyze the evolved coefficients of SAGA, the amount of true positives/negatives and false positives/negatives found by SAGA, and the ability of SAGA to identify the significant features of a problem.

This study begins with a look into past studies of adaptation of EAs. We give a more in depth description of the problem. We describe the workings of SAGA and explain the setup for our study followed by our results. Finally, we analyze the behavior of the SAGA.

2. Related work

For EAs, well-tuned parameters are vital to ensuring good results. Parameter tuning, however, can be a difficult and time consuming task. Thus, how best to handle and simplify parameter tuning is an ongoing problem in the field, with studies dedicated to this task dated from the field's inception through today (De Jong, 1975; Eiben & Smit, 2011a; Mills et al., 2015; Schaffer et al., 1989). This difficulty has even lead to attempts to create a "parameter-less" GA (Harik & Lobo, 1999) which, while greatly improving usability, in practice basically gives default values for the parameters and comes at the cost of lowered final result quality.

One approach to addressing the difficulty of parameter tuning is to use *parameter control*; wherein one or more parameters are dynamic over the course of a run. There are multiple survey papers covering the many different studies and implementations of parameter control that have been proposed over the years (Eiben et al., 1999; Eiben & Smit, 2011b; Hinterding et al., 1997; Karafotias et al., 2015). Parameter control consists of three categories: deterministic, adaptive, and self-adaptive (Eiben et al., 1999). Each category controls the dynamic parameters through different methodologies. Deterministic control is the simplest of the three and changes parameters throughout a run in a fixed, pre-determined manner. Adaptive control changes parameters through some sort of feedback mechanism, e.g. basing a parameter's value on the previous contributions of the parameter or on the population's average fitness. Self-adaptive control encodes the parameters onto the genome of the individuals and evolves parameters using the evolutionary process through which the problem's candidate solutions are evolved.

Deterministic and adaptive control methods often show improved results over fixed parameters; however, they generally do not improve usability as they introduce new mechanisms that themselves must be properly designed or tuned. The most common examples of deterministic control consist of decreasing the mutation rate over time according to some predetermined mathematical formulae (Fogarty, 1989; Hesser & Männer, 1990). This decrease is beneficial as a higher mutation rate early on can aid in exploration, but may be detrimental later in the run when exploitation, the fine-tuning of a good solution, is more important. Selection pressure may also be adjusted over the course of a run to vary the balance of exploitation and exploration (Lu et al., 2015). In practice, deterministic control methods are not commonly used, as adaptive or self-adaptive strategies offer more flexibility.

Adaptive control utilizes a feedback system to intelligently control the dynamics of parameters. The most common approach to adaptive control is to base operator parameters (e.g. crossover rate and mutation rate) on population characteristics such as fitness, diversity, or size (Alsaeedan & Menai, 2015; Castro & Camargo, 2004; Chiba et al., 2019; Contreras et al., 2020; Han & Xiao, 2022; Li et al., 2017; Srinivasa et al., 2007; Sun & Lu, 2019; Thierens, 2002; Venugopal et al., 2009; Yang et al., 2021; Yu et al., 2022; Yuan & Wang, 2019; Zhou et al., 2022), Other examples of adaptive control methods include basing operator rates or step sizes on the fitness contributions of the operators (Lin et al., 2003; Riedel et al., 2005; Xue et al., 2019), adapting parameter values through reinforcement learning (Karafotias et al., 2014), or adjusting crossover exploration proportionally to parent similarity (Deb & Beyer, 2001). Adaptive control methods introduce the possibility of using multiple genetic operator methods, where the algorithm can itself select which operator to use based on the adaptive feedback, such as doing a probabilistic selection for the operator method to use based on the fitness contributions of each operator (Acan et al., 2003; Hadka & Reed, 2013; Hong et al., 2002; Qin & Suganthan, 2005). Studies on adaptive methods also cover aspects of an EA beyond crossover and mutation; such as population size (LaPorte et al., 2015), parent selection pressure (Xie & Zhang, 2013), or the migration interval used in island models (Mambrini & Sudholt, 2015; Srinivasa et al., 2007). While adaptive parameter control methods do remove the need to set some parameters, they instead require the user to design a feedback mechanism, which often has its own parameters that need to be tuned. As a result, the amount of setup required of the user is not lessened, and may even be increased depending on the complexity of the feedback mechanism.

The final parameter control method, self-adaptation, is the approach we use in our work. Of the three types of parameter control, self-adaptation is the most compatible to improving algorithm usability since it does not inherently require the user to design and tune the mechanism in which the parameters are changed, as that is delegated to the evolutionary processes already present within the algorithm. Like adaptive control, self-adaptation can also result in performance improvements; a $(1, \lambda)$ EA is shown to have a lower number of expected evaluations with a self-adaptive mutation rate than with a static rate (Doerr et al., 2018). For GAs, there are multiple studies in the past covering concepts such as self-adaptive mutation rates (Bäck, 1992a, 1992b; Smith & Fogarty, 1996), self-adaptive selection of

Table 1
Self-adaptive algorithm comparison.

Algorithm	Mutation			Crossover		
	Rate	Operator Selection	Operator Arguments/ Step Size	Rate	Operator Selection	Operator Arguments
Bäck (1992b)	X					
Bäck (1992a)	X					
Spears (1995)					X	
Beyer (1996)			X			
Galaviz and Xuri (1996)	X			X		X
Murata and Ishibuchi (1996)		X			X	
Smith and Fogarty (1996)	X					
Hinterding (1997)		X	X			
Greenwood and Zhu (2001)			X			
Hansen and Ostermeier (2001)			X			
Yoon and Moon (2002)					X	
Kivijärvi et al. (2003)	X				X	
Serpell and Smith (2010)	X	X				
Contreras-Bolton and Parada (2015)		X			X	
Doerr et al. (2018)	X					
Sabar et al. (2019)	X	X		X	X	
SAGA	X	X	X	X	X	X

genetic operators (Hinterding, 1997; Murata & Ishibuchi, 1996; Spears, 1995; Yoon & Moon, 2002) or some combination of the two (Galaviz & Xuri, 1996; Kivijärvi et al., 2003; Sabar et al., 2019). CMA-ES, a very effective method for numerical optimization, updates the individuals' mutation step sizes through the self-adaptation of a covariance matrix (Hansen & Ostermeier, 2001). Self-adaptation is also used to dynamically adapt the rules that each individual solution uses to decode their own representation (Hartmann, 2002). The dynamic variability of the self-adaptive parameter can be beneficial since the optimal parameter setting may be different at different points in the execution; furthermore, adaptation and self-adaptation can discover emergent synergy between multiple operations within the algorithm. Some recent approaches examine self-adaptive mutation specifically on permutation representations (Serpell & Smith, 2010) and self-adaptive selection of multiple mutation and crossover operators on the Traveling Salesman Problem (Contreras-Bolton & Parada, 2015). Self-adaptation is an excellent tool for our purposes since it has a history of both improving performance and lessening the burden of parameter tuning. The latter is a primary motivation of this work: to make an EA that is more accessible to domain practitioners who want to apply EAs in their fields.

Table 1 shows how previous works have utilized self-adaptation as compared to our SAGA. The table categorizes the type of self-adaptation according to whether the algorithm adapts the rate, operator selection, or operator arguments for both mutation and crossover. Where our implementation stands out is the scale of the self-adaptation. Previous self-adaptive EA studies apply self-adaptation to only one or a few parameters, focusing on just mutation parameters (Bäck, 1992a, 1992b; Beyer, 1996; Doerr et al., 2018; Greenwood & Zhu, 2001; Hansen & Ostermeier, 2001; Hinterding, 1997; Serpell & Smith, 2010; Smith & Fogarty, 1996), just crossover parameters (Spears, 1995; Yoon & Moon, 2002), just operator type (and not rates) (Contreras-Bolton & Parada, 2015; Murata & Ishibuchi, 1996). In evolutionary strategies, it is the norm to use self-adaptation on the mutation step size (Beyer, 1996; Greenwood & Zhu, 2001; Hansen & Ostermeier, 2001), which we categorize here in the mutation "Operator Arguments/Step Size" column since the step size is controlled through the σ argument of a Gaussian distribution. It should be noted, that in SAGA there is a single mutation step setting per individual while some ES methods have a unique mutation step size for each value in the chromosome (Hansen & Ostermeier, 2001). Due to potential non-linear relationships among system parameters (Grefenstette, 1986; Schaffer et al., 1989), the performance of an EA that self-adapts just a subset of parameters, as well as the self-adaption process of such a system, may be limited by any user specified (non-adaptive) parameters that are set. To maximize the usability improvements of our system, SAGA applies self-adaptation to all of the parameters associated with genetic operators, operator types, and operator rates.

3. Problem

The problem we address is a predictive classification problem in which we use physical therapist (PT) demographic, practice, and market characteristics to predict if the annual Medicare payment to a PT will be above or below the industry median. The large amount of data that is available in this era of big data is a rich source of information for many industries including the health care industry. Medicare is the primary source of health insurance for individuals aged 65 or older in the United States (US). Physical therapy services, which can improve and maintain mobility, strength, and general health, are covered by the Medicare fee-for-service (FFS) program (American Physical Therapy Association, 2014). Due to the aging baby boomer population and to recent declines in birth rate, the 65+ segment of the US population is expected to increase from 15% in 2014 to a projected 24% in 2060 (Mather et al., 2015). As this segment of the US population increases, demand for physical therapy services is also expected to rise. PTs receive payments from Medicare for the clients they serve who are on Medicare. A better understanding of the factors associated with receiving above or below median Medicare payments can inform PTs and policy makers about practice variability as well as provide potential strategies and policies for improving access to physical therapy services for Medicare FFS beneficiaries.

The data that we use to make this classification comes from the 2014 Medicare Provider Utilization and Payment Data: Physician and Other Supplier Public Use File (PUF) and the 2015–2016 Area Health Resources File (AHRF). These data can be split into two groups; provider level variables and county level variables. The provider level variables come from the PUF and give data for 40,662 specific Medicare service providers from across the country. The county level variables come from the AHRF and give data on the local health care markets in which the Medicare service providers reside. Each data point consists of 25 independent variables representing provider and county practice parameters and a corresponding dependent variable representing whether the PT received an above median Medicare payment. Table 2 lists the independent variables and their scopes. Fig. 1 shows a histogram of the distribution of all 40,662 dependent variables of standardized Medicare payments in our data set, along with the overall median value.

4. Algorithm description

We evaluate the performance of SAGA on this problem by comparing it to a canonical GA and to LR a benchmarks. We begin by describing LR. Next, we describe our canonical GA benchmark, the fitness function utilized by both the canonical GA and SAGA, and how SAGA extends the canonical GA.

Table 2
Independent variables.

ID	Independent variable	Level of Observation	Data Type	Notes
1	Gender	Provider	Binary	1 = Female, 0 = Male
2	Doctor of Physical Therapy degree	Provider	Binary	1 = Yes, 0 = No
3	Number of unique HCPCS/CPT codes billed per PT	Provider	Integer	Each HCPCS/CPT code refers to a unique service that may be provided and billed by a PT
4	Number of beneficiaries per PT	Provide	Integer	Number of patients served by the PT that are Medicare beneficiaries.
5	Billed charge to Medicare allowed amount ratio	Provider	Float	Ratio of the total amount billed by the PT for services to the sum of the Medicare allowed amounts for those services.
6	Average Medicare standardized payment per beneficiary	Provider	Float	The total Medicare standardized payment received by a PT divided by the number of unique Medicare beneficiaries treated by the PT.
7	Percent of physical agents	Provider	Float	Proportion of physical agent CPT codes (e.g. hot/code pack, ultrasound, electrical stimulation, etc.) relative to all CPT codes.
8	Percent of therapeutic procedures	Provider	Float	Proportion of therapeutic exercise CPT codes (e.g. manual therapy, therapeutic exercise, physical agents, etc.) relative to all CPT codes.
9	Number of new patients per PT	Provider	Integer	Number of Medicare beneficiaries receiving initial physical therapy evaluation service (CPT code: 97001) from PT.
10	Average age of beneficiaries	Provider	Integer	Average age of PT patients that are Medicare beneficiaries.
11	Average Hierarchical Condition Category (HCC) risk score	Provider	Float	Each Medicare beneficiary is assigned an HCC risk score that predicts the relative cost of treating that beneficiary in the next year, compared to all Medicare beneficiaries. This variable is the average of the risk scores of all beneficiaries treated by the PT.
12	Small metro practice location[a]	Provider	Binary	Practice location is in a metropolitan area with population fewer than 250,000.
13	Mid-sized metro practice location[a]	Provider	Binary	Practice location is in metro area with population of 250,000–1,000,000.
14	Non-metro metro practice location[a]	Provider	Binary	Practice location is outside of a metro area.
15	Standardized risk-adjusted per capita Medicare costs	County	Float	This measure is adjusted to allow for comparison of medical costs in different counties.
16	Primary care physicians per 10,000 population	County	Float	Primary care physicians are those with specialties of general family medicine, general practice, general internal medicine, and general pediatrics.
17	PTs per 10,000 population (2009)	County	Float	Number of PTs per 10,000 population. (The data for this variable is from 2009.)
18	Percent of beneficiaries eligible for Medicaid	County	Float	Percent of Medicare beneficiaries in county who are eligible for Medicaid coverage.
19	Average age of beneficiaries	County	Integer	Average age of Medicare beneficiaries in county.
20	Percent female beneficiaries	County	Float	Percent female Medicare beneficiaries in county
21	Average HCC risk score	County	Float	Average HCC risk score of Medicare beneficiaries in county.
22	Beneficiaries percent of population	County	Float	Percent of Medicare beneficiaries among the population in the county.
23	Median household income	County	Integer	Based on the American Community Survey (ACS)
24	Percent of persons 65 or older in deep poverty	County	Float	Deep poverty refers to households with income that is less than half of their poverty threshold[b].
25	PTs per 10,000 beneficiaries	County	Float	Number of PTs per 10K Medicare beneficiaries in county.

[a]At most one location type will be set to 1. If all binary location types are set to 0, the location type is: Large metro practice location (population size greater than 1,000,000).
[b]The poverty threshold varies based on the size of the family.

4.1. Logistic regression

LR is a commonly used benchmark for predictive classification problems of this kind in the field of health informatics (Chaurasia & Pal, 2014; De Vasconcelos et al., 2001; Min et al., 2006; Serban et al., 2011; Thornblade et al., 2018). LR makes a prediction on whether a data point j belongs to a class x based on the logistic function:

$$p_{x,j} = \frac{1}{1 + e^{-(s_0 + \sum s_i v_{i,j})}} \quad (1)$$

where $p_{x,j}$ is the probability that data point j belongs to class x, s_0 is the intercept, s_i is the coefficient for independent variable i, and $v_{i,j}$ is the independent variable i of data point j. For our implementation, we use the LogisticRegression class from the scikit-learn library in Python.

4.2. Canonical genetic algorithm

We use a canonical generational GA (Grefenstette, 1986; Holland, 1975) as a comparison for SAGA. SAGA extends this canonical GA, thus the purpose of this comparison is to ensure that the changes that are made to a canonical GA to create SAGA do not have a detrimental effect on the outcome.

Algorithm 1 shows the pseudocode for a canonical GA. The algorithm begins with a randomly initialized population of candidate solutions. The GA cycles through multiple generations until a stopping condition is met. In each generation, every individual of the population is evaluated using a fitness function to obtain a measure of the quality of that individual's encoded solution. A selection method probabilistically selects the more fit individuals to become "parents"

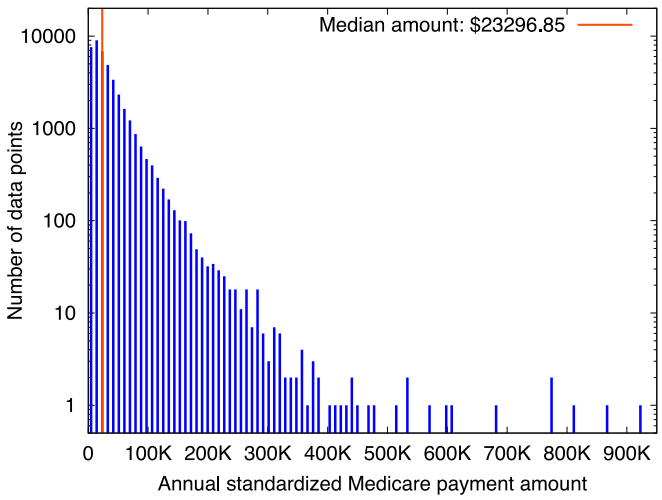

Fig. 1. Histogram of the distribution of the annual standardized Medicare payment amounts. The median value is $23,296.85. Please note that the *y*-axis uses a log scale.

Algorithm 1: Genetic Algorithm.

```
Initialize population of candidate solutions
while stop condition is false do
    Evaluate population (fitness)
    Probabilistically select parents for next generation (selection)
    Apply crossover to parents to generate children (crossover)
    Apply mutation to children (mutation)
    Replace population with children
end while
```

from which new candidate solutions will be created. Genetic operators (crossover and mutation) are applied to the parents to create a new generation of candidate solutions. The new generation of solutions replace the previous generation. Table 3 lists our parameter settings for the canonical GA. These parameter settings are determined empirically.

Each individual or candidate solution in the population consists of a vector of 51 floating-point values, $c_i : i = 0, \ldots, 50$, initialized randomly in the range [-1:1]. The c_0 coefficient encodes a constant while the c_1 to c_{50} coefficients encode values that adjust the impact of the independent variables on the classification outcome. The fitness of an individual is calculated as follows: First, the GA calculates a linear summation, $prediction_j$, for each data point j to be classified:

$$prediction_j = c_0 + \sum_{i=1}^{25} c'_i v_{i,j} \tag{2}$$

where $v_{i,j}$ is the ith independent variable, $i = 1, \ldots, 25$, of the jth data point and c'_i is the ith modified coefficient as calculated by:

$$c'_i = \begin{cases} c_i^1 & \text{if } -1.0 \le c_{i+25} < -0.5 \\ c_i^2 & \text{if } -0.5 \le c_{i+25} < 0.0 \\ c_i^3 & \text{if } 0.0 \le c_{i+25} < 0.5 \\ c_i^4 & \text{if } 0.5 \le c_{i+25} < 1.0 \end{cases} \tag{3}$$

Note that i corresponds with the ID number in column 1 of Table 2. The reason for raising the coefficients c_1 through c_{25} by a power specified by the coefficients c_{26} through c_{50}, respectively, is that the ranges of the independent variables in our data set vary greatly, with some variables having ranges that are magnitudes larger than others. This modification provides a mechanism for dealing with this wide variability of ranges.

Next, we apply the linear summation of Eq. (2) to each of the data points in the training set and determine a classification based on:

$$above\_median_j = \begin{cases} 1 & \text{if } prediction_j > 0.0 \\ 0 & \text{if } prediction_j \le 0.0 \end{cases} \tag{4}$$

Table 3
GA parameters.

Parameter	Value
Stopping Condition	200 generations
Population size	100
Crossover rate	0.9
Crossover operator	two-point
Mutation rate	0.2
Mutation operator	uniform random
Parent Selection method	Tournament, size 10

Algorithm 2: Self Adaptive Genetic Algorithm.

```
Initialize population of candidate solutions
while stop condition is false do
    Evaluate population (fitness)
    Probabilistically select parents for next generation (selection)
    for each set of parents do
        Probabilistically select crossover operator according to encoded weights (see Algorithm 3)
        Apply crossover to parents to generate children
    end for
    for each child do
        Probabilistically select mutation operator according to encoded weights (see Algorithm 3)
        Apply mutation to child
    end for
    Replace population with children
end while
```

Table 4
SAGA Parameters.

Parameter	Value
Stopping Condition	200 generations
Population size	100
Parent Selection method	Tournament, size 10

Both SAGA and the canonical GA use this fitness function.

4.3. Self adaptive genetic algorithm

Accurately tuning EA parameters is known to have a very large impact on the algorithm's performance. Tuning these parameters is also quite difficult and time consuming however, as interdependencies mean that they often cannot be tuned individually. This task can be particularly challenging for someone who is not previously familiar with EAs. SAGA automates the GA parameter tuning process by encoding and evolving many of the parameter values as part of individuals that are evolved in a SAGA population. Specifically, our SAGA implementation self-adapts the parameters of mutation rate, crossover rate, selection of the mutation operator, selection of the crossover operator, and any additional arguments required by the selected mutation and crossover operators. We include a variety of crossover and mutation operators in the list of candidate operators with the goal of covering a range of genetic operator behaviors. Algorithm 2 shows the pseudocode for SAGA. Table 4 gives the basic parameter settings that we use for SAGA on this problem.

The addition of self-adaptation adds new information that must be encoded within an individual. SAGA representation takes inspiration from previous EA setups that utilize multiple chromosomes (Cavill et al., 2005; Hinterding, 1997; Kühn et al., 2013) and consists of two chromosomes of data. Fig. 2 shows the SAGA representation. The first chromosome is called the *solution chromosome* and encodes the same information as that encoded by an individual in the canonical GA. The second chromosome is called the *parameter chromosome* and encodes all of the self-adaptive parameter information. The values in

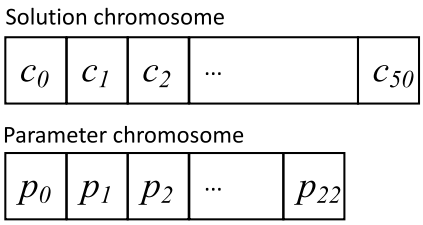

Fig. 2. SAGA uses a multi-chromosome representation with one chromosome encoding the solution and one chromosome encoding parameter values.

Table 5
SAGA Crossover Operators.

Crossover operator	Number of Arguments
Two-point	0
Simulated binary (Deb & Agrawal, 1995)	1
Blend (Eshelman & Schaffer, 1993)	1
Arithmetic (Michalewicz, 1992)	1
Linear (Wright, 1991)	0
Simplex (Tsutsui et al., 1999)	1
UNDX (Kita et al., 2000)	2
PCX (Deb et al., 2002)	2

Table 6
SAGA Mutation Operators.

Mutation operator	Number of Arguments
Uniform random	0
Gaussian	1
Polynomial (Deb & Deb, 2014)	1

the parameter chromosome are modified over the course of execution using the same evolutionary mechanism that modifies the solution chromosome. The modification of these parameter chromosome values thus modifies the very evolutionary mechanisms themselves for the subsequent generations.

The parameter chromosome consists of a vector of 23 floating-point values, $p_i : i = 0, \ldots, 22$, initialized randomly in the range [0,1]. The size of the parameter chromosome depends on the number of crossover and mutation operators employed; our implementation includes eight crossover operators and three mutation operators. Tables 5 and 6 give the crossover and mutation operators, respectively, employed by SAGA along with the number of arguments each operator requires. The elements specified in the parameter chromosome include three types of values: operator rates, operator fitnesses, and operator arguments. Table 7 lists the specific elements encoded in the parameter chromosome.

Two values in the parameter chromosome, p_0 and p_6, specify the self-adaptive mutation and crossover rates, respectively. Each encoded value represents the associated rate for that individual. For mutation, each individual utilizes its own encoded mutation rate. For crossover, the effective crossover rate of a pairing of parents is the average of the parents' crossover rates.

Eleven values in the parameter chromosome specify *operator fitness* values. SAGA employs multiple operators for mutation and crossover, unlike a canonical GA which has a single operator for each. Every time mutation or crossover is to occur, one of the employed operators is probabilistically selected. To facilitate this selection, each of the candidate mutation/crossover operators is assigned an operator fitness value, a floating-point value in the range of [0,1] that is encoded in the parameter chromosome. With regard to the probabilistic selection mechanism, operator fitnesses are to the crossover/mutation operators what fitness is to the individual; a higher value improves the probability of an operator being selected. The idea is that the more effective operators should evolve to have higher operator fitnesses and therefore be used more often than the less effective operators.

Table 7
Parameter chromosome specification.

Value	Operator	Operator Fitness	Operator Argument
p_0	Mutation rate		
p_1	Gaussian mutation	X	
p_2	Gaussian mutation		X
p_3	Polynomial mutation	X	
p_4	Polynomial mutation		X
p_5	Uniform Random mutation	X	
p_6	Crossover rate		
p_7	Two-point crossover	X	
p_8	Simulated Binary crossover	X	
p_9	Simulated Binary crossover		X
p_{10}	Blend crossover	X	
p_{11}	Blend crossover		X
p_{12}	Arithmetic crossover	X	
p_{13}	Arithmetic crossover		X
p_{14}	Linear crossover	X	
p_{15}	Simplex crossover	X	
p_{16}	Simplex crossover		X
p_{17}	PCX crossover	X	
p_{18}	PCX crossover		X
p_{19}	PCX crossover		X
p_{20}	UNDX crossover	X	
p_{21}	UNDX crossover		X
p_{22}	UNDX crossover		X

Algorithm 3: SAGA Crossover/Mutation Selection Process.

```
Order crossover/mutation operators by operator fitness (high to low)
for each crossover/mutation operator in ordered list do
    if not last operator in list then
        Generate random number in [0,1]
        if Random number less than 0.9 then
            Select operator and exit for loop
        else
            Move to next operator in ordered list
        end if
    else if last operator in list then
        Select operator and exit for loop
    end if
end for
```

Ten values in the parameter chromosome specify *operator argument* values. Some genetic operators require one or more arguments. For example, Gaussian mutation requires a sigma argument to determine the size of the Gaussian distribution. All operator arguments are encoded as floating-point values in the range of [0,1] within the parameter chromosome. If the argument in question has a range that is not [0,1], then the encoded value in the range of [0,1] is mapped to the expected range of the argument when the argument is utilized.

The SAGA crossover and mutation operator selection processes probabilistically favor operators with higher operator fitness values. Although crossover selection and mutation selection are separate processes, both follow the logic presented in Algorithm 3. For the selection of a mutation operator, each individual utilizes the mutation operator fitnesses encoded in its own parameter chromosome. For the selection of a crossover operator, the effective operator fitnesses for each of the crossover operators is the average of the parents' encoded crossover operator fitnesses. Because crossover mixes the encoded data of multiple parent individuals, how crossover handles multiple chromosomes of data must be addressed. In SAGA, when crossover occurs, the parents' solution chromosomes mix with each other and the parameter chromosomes mix with each other; there is no crossover between solution and parameter chromosomes.

Fig. 3 illustrates the behavior of each of the crossover operators that we include in SAGA. Each of these illustrations shows a two-dimensional plot for a theoretical setup where each individual contains

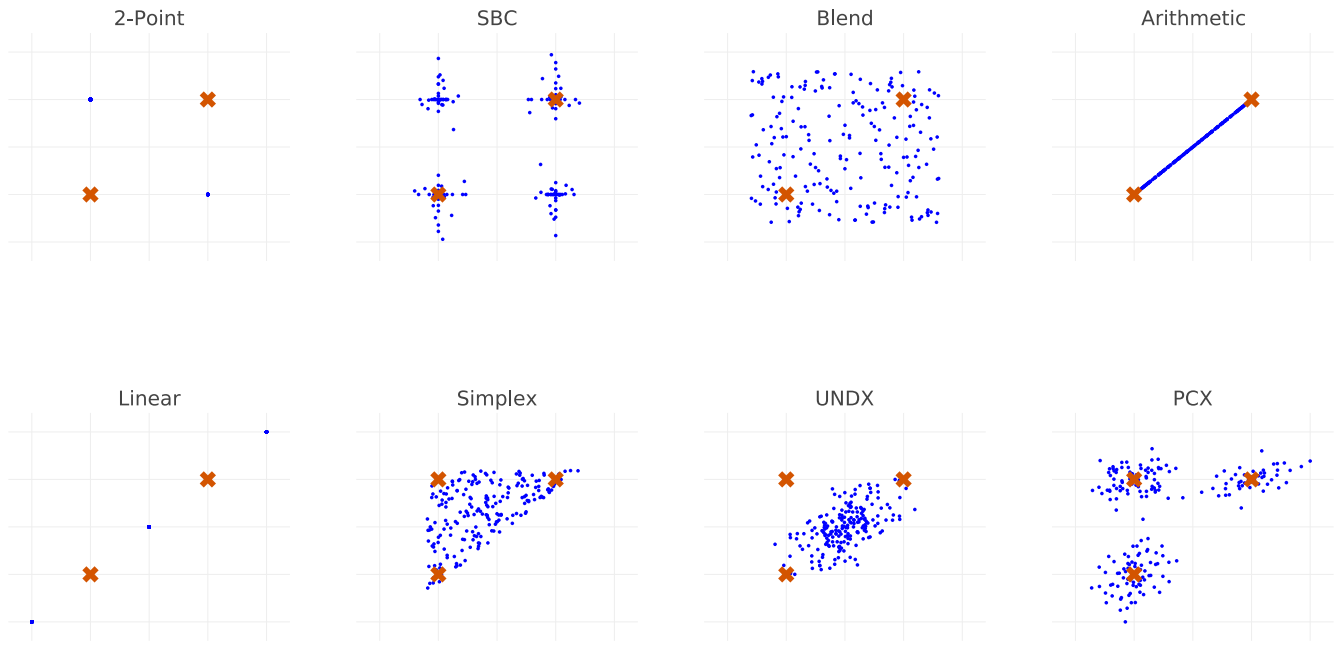

Fig. 3. SAGA Crossovers: Examples of the space of 100 potential offspring (designated by dots) generated from the crossover of a set of fixed parents (designated by X).

only two values in a chromosome and each axis represents one of these values. Parents are represented by X and 100 children, represented by dots, are randomly generated by applying crossover to the parents. These plots show that there is a wide variety of possible crossover behaviors; we chose these crossover operators to purposefully have a diversity of behaviors so that the SAGA can adapt to whichever is most beneficial. Note that some of these operators use three parents instead of the traditional two. We do not give detailed mathematical descriptions of each of these crossover operators as some are fairly complex and require extensive explanation. Tables 5 and 6 provide references to previous works that do contain the detailed descriptions for each operator if such is desired.

5. Experimental methods

We evaluate the performance of the SAGA on the problem of classifying if the Medicare standardized payment for physical therapists is above or below the national average and compare its performance to that of LR as a benchmark. We also compare the SAGA's performance to that of a canonical GA to verify that SAGA can perform at least as well as a canonical GA. Our evaluation metric is the percentage of correct classifications. We split the input data into a training set and a test set, where the algorithm is trained on only the training set but is evaluated against both. We run the problem on five data sets, each consisting of a different ratio of training and test data; these training:test ratios are: 50:50, 25:75, 10:90, 5:95, and 1:99. The training set sizes are no larger than 50% because a previous study (Wu et al., 2019) showed no significant difference in results for this same problem while using training sets ranging from 50% to 85%.

We run the SAGA and GA on both the raw input data and standardized input data. The ranges of the independent variables vary greatly and standardization provides a mechanism for equalizing these ranges such that the larger range variables do not have an artificially large impact on the result. Although we do give the SAGA and GA a mechanism for adjusting the magnitude of the coefficients, we do not know how effectively that mechanism will be used by the SAGA and GA. Testing the SAGA and GA on both standardized and raw data will give us an indication of how well the SAGA and GA can evolve coefficient magnitudes. We therefore test five methods: SAGA on raw data (RD-SAGA), SAGA on standardized data (SD-SAGA), GA on raw data (RD-GA), GA on standardized data (SD-GA), and LR.

The standardized independent variables are calculated as follows. Let $v'_{i,j}$ represent the standardized value of the ith independent variable from data point j. Then

$$v'_{i,j} = \frac{v_{i,j} - \vec{v}_i}{\sigma_i} \tag{5}$$

where $v_{i,j}$ is the corresponding raw independent variable,

$$\vec{v}_i = \frac{1}{40662} \sum_{r=1}^{40662} v_{r,i} \tag{6}$$

is the average of the ith independent variable across all 40,662 data points, and σ_i is the standard deviation of $\vec{v}_i$.

6. Results

Our results evaluate performance with respect to two evaluation measures. The first and primary measurement is accuracy; the number of correct classifications. The second measurement is a breakdown of the accuracy into the number of true positive, true negative, false positive, and false negative classifications.

Tables 8 and 9 show the accuracy results of our study on the training and test data sets, respectively. The tables show the accuracy of each of the algorithms for each data set. We execute 50 runs of the SAGA and GA algorithms, due to their pseudo-random nature; therefore, the tables show the best single run and the average of all 50 runs, along with the 95% confidence interval. In addition, the test data table, Table 9, highlights the best accuracy in bold.

SD-SAGA shows the best overall accuracy across the data sets, outperforming LR and the other SAGA and GA methods on both test and training sets. The only instance in which SD-SAGA is not the best performer is the 99% test data set, in which RD-GA slightly outperforms it by 0.15%. The RD-GA appears to be the second best performer overall, followed the RD-SAGA, LR, and lastly SD-GA.

Interestingly, the methods that use standardized data are both the best and worst performing of the SAGA and GA methods. In the case of SAGA, standardization provides a clear advantage, with SD-SAGA outperforming RD-SAGA on every instance, while the opposite is true

Table 8
Training data accuracy results. The Best columns show the result of the highest accuracy run and the average columns show the average accuracy of all 50 runs along with the 95% confidence interval.

Data set	RD-SAGA		SD-SAGA		RD-GA		SD-GA		LR
	Best	Avg	Best	Avg	Best	Avg	Best	Avg	
Train 50%	93.10	77.82 ± 3.79	94.17	93.45 ± 0.16	93.26	89.37 ± 0.96	91.37	90.46 ± 0.16	93.11
Train 25%	93.65	80.89 ± 3.42	94.14	93.58 ± 0.15	93.25	89.87 ± 0.77	91.53	90.27 ± 0.17	92.95
Train 10%	93.33	83.52 ± 2.88	94.45	93.00 ± 0.55	92.97	90.00 ± 0.66	91.69	90.37 ± 0.13	92.94
Train 5%	94.10	81.38 ± 3.26	95.47	93.86 ± 0.58	94.44	90.14 ± 0.91	93.51	91.14 ± 0.17	93.85
Train 1%	94.34	77.63 ± 3.31	96.06	92.16 ± 0.75	93.10	88.53 ± 0.87	91.87	90.12 ± 0.21	92.86

Table 9
Test data accuracy results. The Best columns show the result of the highest accuracy run and the average columns show the average accuracy of all 50 runs along with the 95% confidence interval. The best accuracy across the five methods is bolded.

Data set	RD-SAGA		SD-SAGA		RD-GA		SD-GA		LR
	Best	Avg	Best	Avg	Best	Avg	Best	Avg	
Test 50%	92.92	77.80 ± 13.61	**93.84**	93.15 ± 0.54	93.12	89.35 ± 0.96	91.32	90.40 ± 0.17	92.90
Test 75%	93.20	80.92 ± 12.47	**93.86**	93.25 ± 0.49	93.15	90.08 ± 0.74	91.44	90.40 ± 0.17	93.02
Test 90%	93.23	83.06 ± 10.56	**93.80**	92.40 ± 1.92	93.47	89.82 ± 0.69	91.85	90.21 ± 0.17	92.93
Test 95%	92.76	80.09 ± 11.68	**93.40**	91.91 ± 1.89	93.17	89.16 ± 0.87	91.61	90.07 ± 0.20	93.01
Test 99%	91.74	75.60 ± 12.14	91.95	87.92 ± 3.02	**92.10**	86.23 ± 0.97	91.45	87.90 ± 0.46	90.98

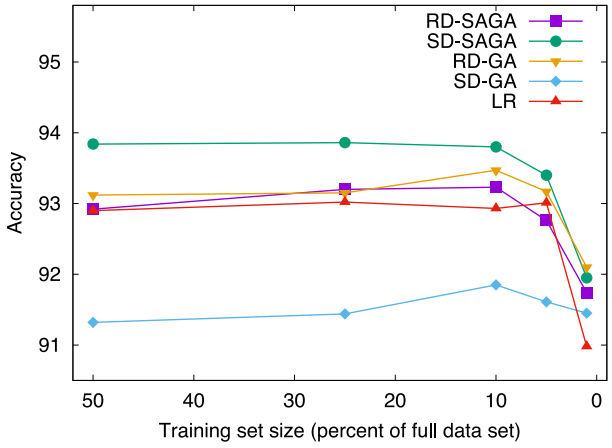

Fig. 4. Best accuracy achieved for each training set size (percent of full data set).

for the GA methods. One consistent effect standardization has on both SAGA and GA is that it lowers the variation from run to run, as shown by the smaller confidence intervals and smaller distance between average accuracy and best accuracy on both of the standardized data methods in Tables 8 and 9. Increased consistency across runs is neither necessarily a benefit nor a detriment in and of itself. Increased consistency can mean that the GA/SAGA more consistently converges at the optimal (or near optimal) solution, or it can mean that it more consistently converges prematurely, on a sub-optimal solution.

Fig. 4 shows how accuracy changes with decreasing training set size. The x-axis plots training set size and the y-axis plots accuracy achieved on the test data set. Results indicate that there is little to no degradation in the classification accuracy on test data set as the training set size decreases from 50% down to 10% of the input data. Only when the training set is down to 1% of the input data, equal to around 400 data points, is there a clear and consistent drop in accuracy on test data across the algorithms when compared to all other training set sizes. On this particular problem, it appears that large training data sets are not required.

Table 10 gives the number of true positive (TP), true negative (TN), false positive (FP), and false negative (FN) classifications accrued on the test set experiments, where a positive classification represents a Medicare standardized payment that is above the median. In each column the best result is in bold. For TP and TN, the best result is the largest value. For FP and FN, the best result is the smallest value. In the case of the SAGA and GA methods, these results refer to the best run as shown in Table 9.

Table 10
Statistical results of test data sets. Best results are in bold.

50:50 test set	TP	TN	FP	FN
RD-SAGA	9698	9231	983	419
SD-SAGA	**9829**	9211	906	**348**
RD-GA	9647	9285	832	567
SD-GA	9352	9215	902	862
LR	9424	**9470**	**647**	790
25:75 test set	**TP**	**TN**	**FP**	**FN**
RD-SAGA	**14625**	13798	1424	**649**
SD-SAGA	14559	13982	1240	715
RD-GA	14540	13867	1355	734
SD-GA	13952	13935	1287	1322
LR	14120	**14232**	**990**	1154
10:90 test set	**TP**	**TN**	**FP**	**FN**
RD-SAGA	17672	16447	1845	**631**
SD-SAGA	17643	16604	1688	660
RD-GA	**17646**	16558	1734	657
SD-GA	16620	16991	1301	1683
LR	16870	**17151**	**1141**	1433
5:95 test set	**TP**	**TN**	**FP**	**FN**
RD-SAGA	18707	17005	2293	623
SD-SAGA	18339	17568	1730	991
RD-GA	**18818**	17172	2126	**512**
SD-GA	17753	17635	1663	1577
LR	17990	**17932**	**1366**	1340
1:99 test set	**TP**	**TN**	**FP**	**FN**
RD-SAGA	19046	17882	2245	1082
SD-SAGA	18976	17802	2325	1152
RD-GA	18724	18351	1776	1404
SD-GA	**19196**	17616	2511	**932**
LR	18208	**18406**	**1721**	1920

Table 10 has a few noteworthy observations. First, is that despite having the best accuracy in all but the 99% test set, SD-SAGA rarely scores the best in any one category of TP, TN, FP, or FN. Rather, SD-SAGA begets the best overall accuracy by getting good enough results on each of the four categories. Another observation is that the SAGA and GA methods tend to favor positive classifications, having a larger ratio of TP to TN and FP to FN. The only exception is the SD-GA which

produces a more even ratio of positive and negative classifications on all but the 99% test set. LR, on the other hand, is much more balanced in regards to the amount of positive and negative classifications; in fact, LR consistently has the lowest number of false positives in all test sets. This leads to LR having the best record for correctly classifying negative classifications. The SAGA and GA methods however are better at avoiding incorrect negative classifications, leading to higher correct positive classifications. Because the dividing point between our two classes is the median of the medicare standardized payments, and a median is by definition in the middle, our data is evenly split between the two classes. This means that LR is actually correct in making roughly equal positive and negative classifications; however, in total, LR makes more incorrect classifications. Hence, the SAGA methods, despite disproportionately favoring positive classifications, return higher overall accuracies.

7. Analysis

We take a closer look at the results of SAGA in order to better understand why it evolves as it does. While our focus is on SAGA, we similarly examine the GA to see if the self-adaptive mechanisms cause any noticeable divergences in evolutionary behavior when compared to a canonical GA. We begin by examining the evolved coefficient values to see if they provide any insight on how the independent variables are used to make a classification. To help with this analysis, we examine how the characteristics of the independent variables from the input data relate to the coefficients. Our observations suggest that the SAGA and the GA may be performing feature selection in conjunction with finding appropriate coefficient values, so we further explore the ability of SAGA and GA to effectively enact feature selection during evolution.

7.1. Coefficient analysis

Here we take a deeper look into the evolved coefficients. Figs. 5 and 6 are both organized in the same manner. Fig. 5 shows the evolved coefficients from the best individual of every run for both RD-SAGA and RD-GA. Fig. 6 shows the same for SD-SAGA and SD-GA. The y-axis displays the value of the coefficient and the x-axis shows the ID number of the independent variable to which each coefficient corresponds. There are 50 lines in each coefficient group along the x-axis corresponding to each of the 50 runs. The x-axis starts with a 0th value; this is the intercept and is marked as 0 so that the other coefficient numbers align properly with the IDs in Table 2.

First, we see that there is a larger difference between standardized data and raw data than there is between SAGA and GA. The evolved coefficients are very similar for SD-SAGA and SD-GA and the evolved coefficients are also very similar for the RD-SAGA and RD-GA. This consistent similarity suggests that the self-adaptive mechanisms that are added to SAGA to dynamically control genetic operator parameters do not significantly change the evolution of candidate solutions as compared to a GA.

The two raw data methods, RD-SAGA and RD-GA, (Fig. 5) both show very inconsistent and chaotic results. For the majority of the independent variables, the evolved coefficients are all over the place. Not only are the magnitudes of the coefficients very different from run to run, but the signs are as well; it is not uncommon to see runs with coefficients near both extremes of 1 and −1 for the same independent variable. Despite this noise, there are a few independent variables that are evolved to consistent values in almost every run; four, six, nine, fifteen, and twenty-three, which according to Table 2 correspond to "Number of beneficiaries", "Average Medicare standardized payment per beneficiary", "HCC risk score", "Primary care physicians per 10,000 population (county)", and "Median household income (county)". Four, six, and nine are consistently positive, usually with a medium to large magnitude while fifteen and twenty-three are consistently near zero. Variables four, six, and nine appear to be the most important in determining the outcome while fifteen and twenty-three appear to be disruptive and are thus evolved to zero to remove their influence entirely; this is further reinforced when we analyze both the coefficients of SD-SAGA and SD-GA and the input data.

The plots for the standardized data methods, SD-SAGA and SD-GA, in Fig. 6 show more consistent coefficients as compared to RD-SAGA and RD-GA. Similar to RD-SAGA and RD-GA, we see that the coefficients at indices four, six, and nine all have highly positive magnitudes, signaling their significance. These three coefficients are even more consistent here than in the raw data methods, and nearly always evolve to magnitudes near the maximum of 1. The variables that consistently evolve to zero in the raw data methods, fifteen and twenty-three, are again almost always very near zero; however, they no longer stand out, as nearly all other coefficients are similarly evolving to zero.

Furthermore, the standardized data methods drastically reduce the magnitude of the majority of the coefficients. This is especially true on SD-SAGA, where most coefficients evolve to near zero. By evolving coefficients near zero, SD-SAGA is effectively removing the associated independent variable from the calculation, since any number multiplied by a sufficiently small number will be nearly zero. Thus, SD-SAGA and SD-GA are effectively enacting simultaneous feature selection and classification through its evolutionary mechanisms. This is less true on the smaller training set sizes, where apparently there is not enough information to fine tune these coefficients as effectively; the lower accuracy on the 99% test set is most likely due to this. Further exploration of this idea of GAs for simultaneous feature selection and classification is worth investigating in the future.

7.2. Input analysis

We need more background information to understand why the coefficients evolved as they do, so we take a look at the input data and its relation to the evolved coefficients. Fig. 7 shows histograms of each of the independent variables. The x-axes indicate the value of the independent variable and the y-axes show the frequency of that value. These plots are generated from the full input data set of 40,662 data points.

First, we can see why standardization can have a stabilizing effect on the evolution. There is a very large difference in the ranges of these variables; some are binary, 0 or 1, some are continuous with very small ranges, from 0 to less than 1, and others still are continuous from 0 up to tens or hundreds of thousands.

All of the independent variables have non-negative ranges, which further explain some of the differences between the raw data and standardized data methods. Thus, when running on the raw data, to push towards a negative classification, there must be some negative coefficients or a negative intercept; this can be observed in Fig. 5. The process of standardization transforms the data such that there are both positive and negative inputs. Thus, negative coefficients are no longer required to obtain negative classifications, and in fact the plots of Fig. 6 show nearly exclusively positive coefficients and intercepts.

We identify five independent variables that have ranges larger than [0,100]: four, six, nine, fifteen, and twenty-three. Without standardization, these five variables will have the largest impact on the outcome because of their larger ranges. These five variables match perfectly with the five variables identified earlier that are the only consistent variables on the raw data methods. This explains the inconsistent and chaotic nature of the coefficients on the remaining variables on RD-SAGA and RD-GA; the other variables are too small to have a significant effect on the outcome and so their coefficient does not matter.

We identify variables four, six, and nine as the influential variables for making a correct classification since they are important on both the standardized data and raw data methods. These three variables are also among the five identified as having larger ranges. This observation leads to the conclusion that the raw data methods benefit from the fact that these influential variables have large ranges. If the opposite

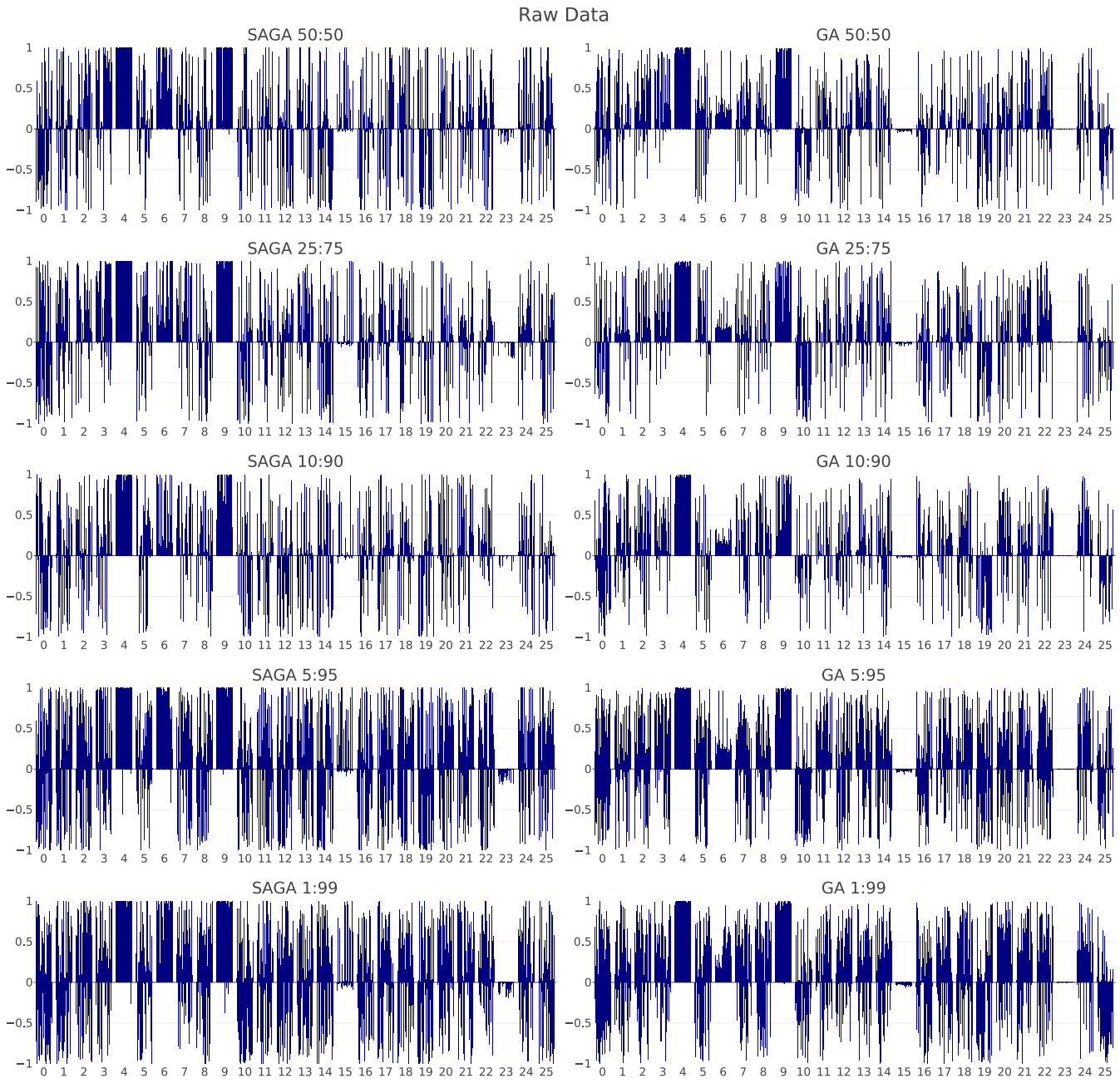

Fig. 5. Best coefficients for RD-SAGA (left column) and RD-GA (right column) on all 50 runs. The training:test ratios are indicated on each plot.

is true, such that the influential variables have small ranges and the non-influential variables have large ranges, then RD-SAGA and RD-GA would have had much more difficulty. This theoretical situation, however, does not guarantee that RD-SAGA and RD-GA will perform poorly; it would require these algorithms to evolve all the large range, non-influential coefficients to near zero and still evolve large coefficients on the small range, influential variables. In conclusion, despite the fact that RD-SAGA and RD-GA perform well here, GA methods, self-adaptive or otherwise, should follow the norms of other classification ML algorithms in standardizing their input data.

7.3. Significant variables analysis

In order to further test the simultaneous feature selection and classification abilities of SAGA, we run the problem two more times; first using only the significant independent variables as identified earlier (4, 6, 9) and second using all but those three independent variables. With this extra experimentation, we want to see how correct SAGA is in its identification of the three significant variables (4, 6, 9). Do the SAGA and GA methods focus on these three because they are required for high accuracy, or can similar accuracy be reached without them? We also include LR again to continue its role as a comparison benchmark.

Tables 11 and 12 show the results of the experiment when using only the three significant variables (4, 6, 9) on training and test data, respectively. Similarly, Tables 13 and 14 show the results when using all variables except the three significant variables on training and test data, respectively. Comparing the results of these tables, we can clearly see that every method scores significantly higher when using the significant variables, with accuracies that are 20% to 30% higher across the board. The insignificant independent variables alone are not sufficient to accurately classify this problem through either SAGA, GA, or LR. These results validate the SAGA's capabilities for simultaneous

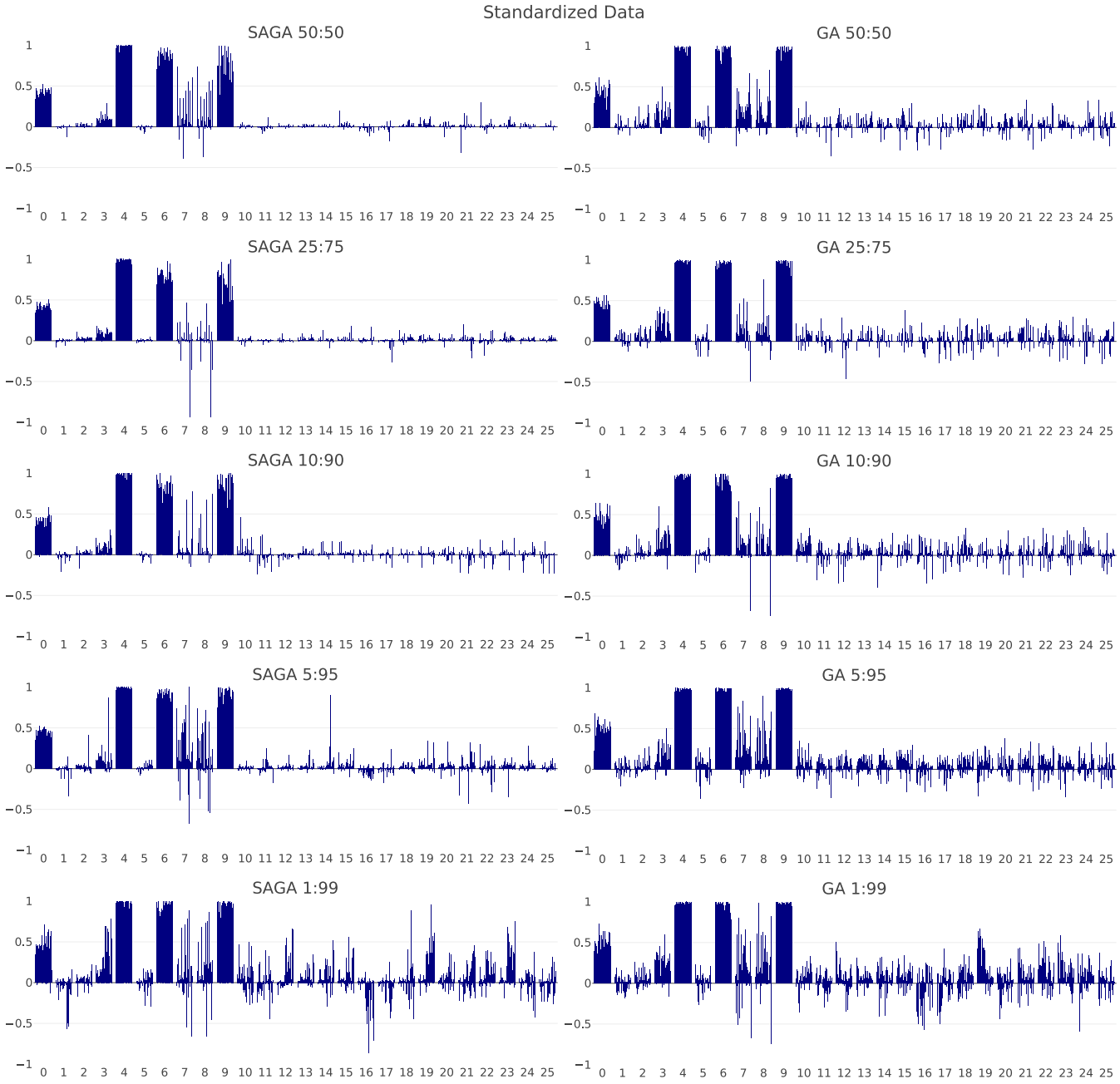

Fig. 6. Best coefficients for SD-SAGA (left column) and SD-GA (right column) on all 50 runs. The training:test ratios are indicated on each plot.

feature selection and classification on this data set, as the variables deemed significant by SAGA are indeed the only variables that are necessary to finding high accuracy results.

Using only significant variables has a large improvement on the performance of SD-GA, no change in performance for SD-SAGA, and a slight improvement on the remaining three methods. SD-GA gains 2.48% in performance from 91.32% in Table 9 to 93.80% in Table 12 on the 50% test data set, with similar improvements on the other data sets. The same improvement is not made on SD-SAGA, which is the best performance method when all variables are used. Using only significant variables results in an SD-SAGA performance of 93.82% on the 50% test data set, which is nearly identical to the 93.84% performance with all variables from Table 9. RD-SAGA, RD-GA, and LR all gain 1% or less on most of the data sets. The 99% test data set is the one exception, where RD-SAGA, RD-GA, and LR gain around 2% when using only significant variables. The results are also somewhat equalized across the methods, with multiple ties for best accuracy across the SAGA and GA methods. These results show that SD-SAGA is the most capable at handling the insignificant variables; reinforced by the coefficient figures in Section 7.1, where SD-SAGA finds the most consistent coefficients for the insignificant variables. The inclusion of more variables increases the search space, increasing the difficulty of the problem; this increase in difficulty is not a factor for SD-SAGA, which is effective enough to find excellent results on both sets of variables.

Fig. 8 shows the plots of the evolved coefficients on the 50:50 training:test set using only the significant variables. We see a very large difference between the evolved weights of the standardized data methods and raw data methods for both SAGA and GA. On the raw data, the intercept is negative, coefficient 4 is very small in either direction, coefficient 6 is very small and negative, and coefficient 9 is positive with either a very small or very large magnitude. Closer

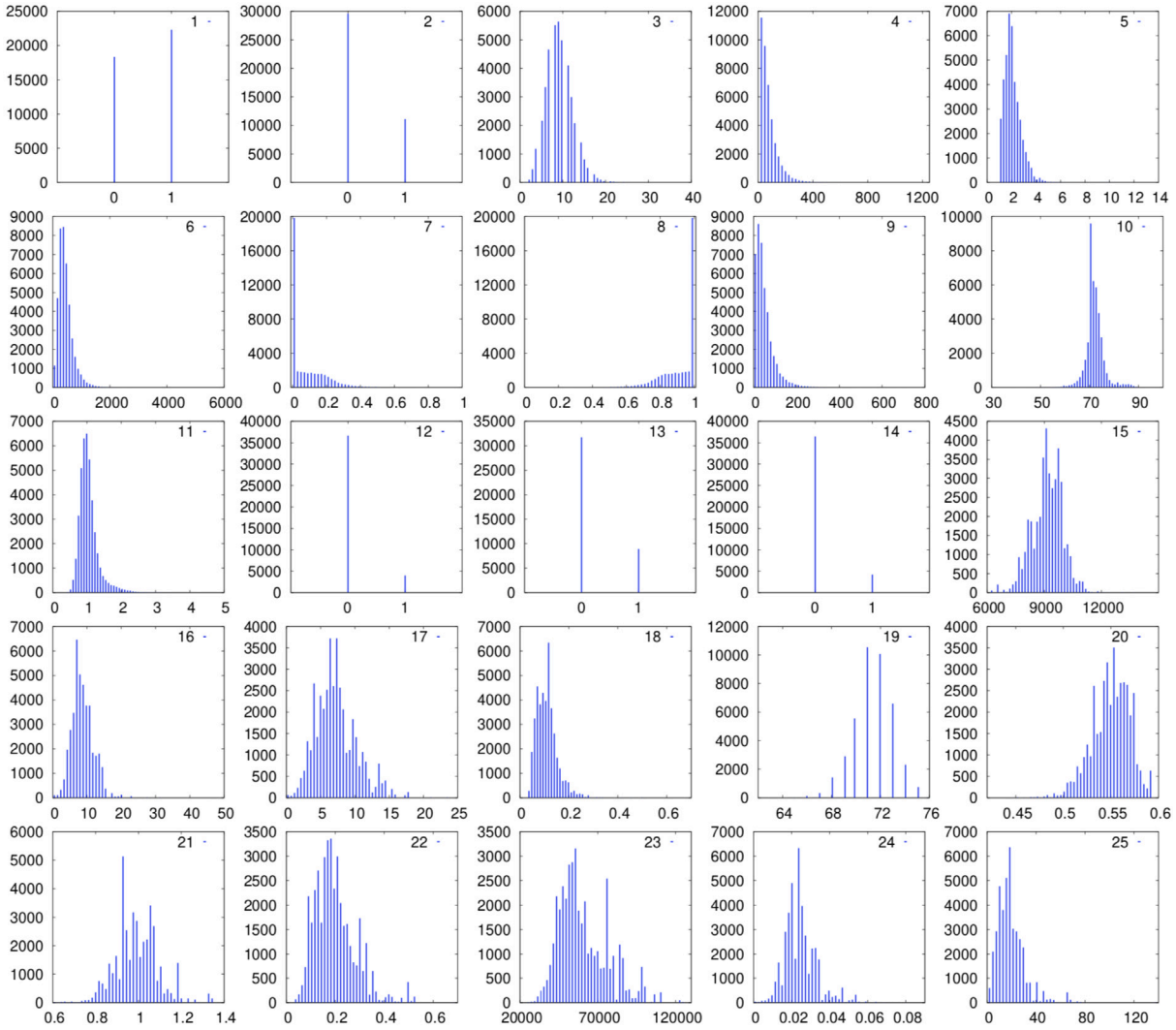

Fig. 7. Histograms of the values of each independent variable. The variable ID numbers in the top right corner of each subplot correlate with the ID numbers in the leftmost column of Table 2.

Table 11
Training data accuracy results using only significant variables. The Best columns show the result of the highest accuracy run and the average columns show the average accuracy of all 50 runs along with the 95% confidence interval.

Data set	RD-SAGA		SD-SAGA		RD-GA		SD-GA		LR
	Best	Avg	Best	Avg	Best	Avg	Best	Avg	
Train 50%	94.09	83.30 ± 3.06	94.11	94.04 ± 0.03	94.00	91.94 ± 1.22	94.09	94.07 ± 4.0e−5	93.15
Train 25%	93.99	82.04 ± 2.98	94.03	93.98 ± 0.01	93.93	90.21 ± 1.99	94.03	93.99 ± 4.9e−5	93.02
Train 10%	94.05	83.23 ± 2.95	94.10	93.90 ± 0.04	93.80	90.52 ± 1.79	94.10	94.00 ± 0.03	92.75
Train 5%	94.88	76.22 ± 3.18	94.98	94.81 ± 0.04	94.84	91.98 ± 1.70	94.98	94.86 ± 0.02	94.20
Train 1%	95.07	78.85 ± 3.62	95.07	94.86 ± 0.08	94.83	91.70 ± 1.81	95.07	94.99 ± 0.04	91.40

Table 12
Test data accuracy results using only significant variables. The Best columns show the result of the highest accuracy run and the average columns show the average accuracy of all 50 runs along with the 95% confidence interval. The best accuracy across the five methods is bolded.

Data set	RD-SAGA		SD-SAGA		RD-GA		SD-GA		LR
	Best	Avg	Best	Avg	Best	Avg	Best	Avg	
Test 50%	**93.84**	83.74 ± 2.94	93.82	93.69 ± 0.03	93.72	91.88 ± 1.17	93.80	93.70 ± 0.01	92.92
Test 75%	93.92	82.04 ± 2.99	93.92	93.82 ± 0.01	93.87	90.10 ± 1.98	**93.94**	93.82 ± 0.01	93.02
Test 90%	93.80	82.86 ± 2.98	93.88	93.67 ± 0.05	**93.89**	90.48 ± 1.83	93.86	93.75 ± 0.03	93.12
Test 95%	93.76	78.35 ± 3.18	**93.90**	93.70 ± 0.05	93.82	91.18 ± 1.70	**93.90**	93.73 ± 0.04	93.28
Test 99%	**93.85**	80.07 ± 3.14	93.49	92.19 ± 0.14	93.83	91.05 ± 1.62	92.64	92.26 ± 0.07	92.92

Table 13
Training data accuracy results using only insignificant variables. The Best columns show the result of the highest accuracy run and the average columns show the average accuracy of all 50 runs along with the 95% confidence interval.

Data set	RD-SAGA		SD-SAGA		RD-GA		SD-GA		LR
	Best	Avg	Best	Avg	Best	Avg	Best	Avg	
Train 50%	67.38	56.09 ± 0.91	71.76	71.20 ± 0.08	65.09	55.83 ± 0.63	71.02	70.57 ± 0.04	69.84
Train 25%	66.97	55.36 ± 0.57	71.53	70.91 ± 0.11	65.17	55.78 ± 0.66	70.52	70.27 ± 0.04	69.07
Train 10%	61.29	55.49 ± 0.39	71.77	70.87 ± 0.14	66.01	57.97 ± 1.10	70.59	70.31 ± 0.04	68.85
Train 5%	62.03	55.03 ± 0.30	72.21	71.08 ± 0.20	64.39	55.96 ± 0.66	71.08	70.60 ± 0.05	67.60
Train 1%	62.81	56.50 ± 0.37	77.59	75.07 ± 0.43	66.50	57.49 ± 0.64	75.62	74.91 ± 0.09	69.78

Table 14
Test data accuracy results using only insignificant variables. The Best columns show the result of the highest accuracy run and the average columns show the average accuracy of all 50 runs along with the 95% confidence interval. The best accuracy across the five methods is bolded.

Data set	RD-SAGA		SD-SAGA		RD-GA		SD-GA		LR
	Best	Avg	Best	Avg	Best	Avg	Best	Avg	
Test 50%	63.12	49.62 ± 1.25	**70.88**	70.23 ± 0.10	65.51	51.38 ± 0.80	70.58	70.02 ± 0.08	69.76
Test 75%	63.91	50.73 ± 1.40	**71.09**	70.42 ± 0.11	60.16	50.46 ± 1.15	70.67	70.28 ± 0.06	69.21
Test 90%	63.26	50.45 ± 0.65	**70.72**	69.98 ± 0.14	65.68	51.56 ± 1.27	70.61	69.99 ± 0.09	68.99
Test 95%	66.85	49.94 ± 1.11	**70.63**	69.26 ± 0.18	59.63	50.47 ± 0.97	70.25	69.49 ± 0.11	68.80
Test 99%	59.38	49.88 ± 1.04	**68.98**	66.92 ± 0.32	65.19	50.18 ± 1.47	68.63	67.21 ± 0.22	67.96

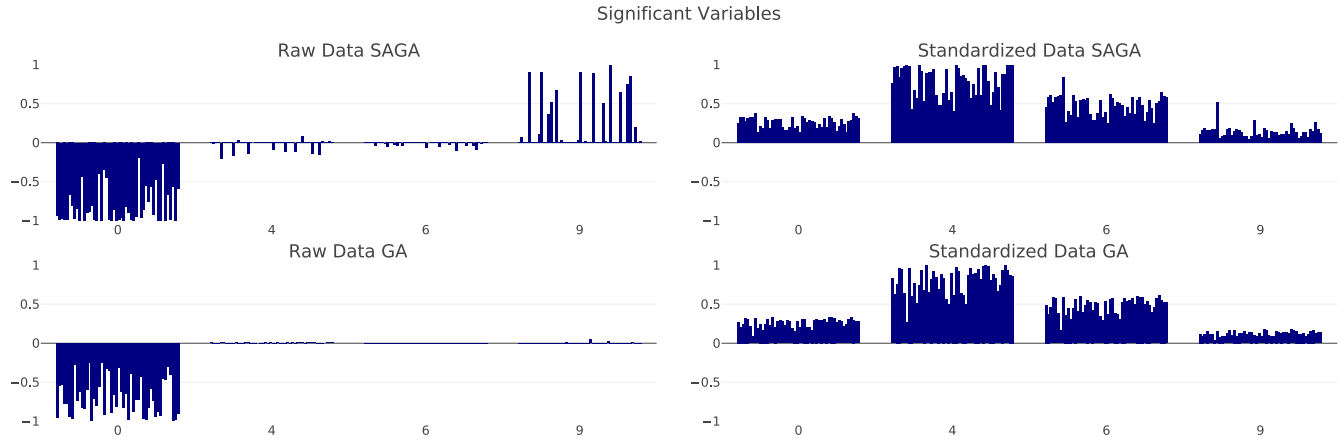

Fig. 8. GAs and SAGA coefficients using only the significant variables on the 50:50 training:test set.

inspection reveals that RD-SAGA finds two different coefficient configurations that are viable; in one, both coefficient 4 and 9 are slightly positive. In the other configuration, coefficient 4 is slightly negative, and coefficient 9 is highly positive. Both of these configurations are very different from the more consistent results of the standardized data methods, wherein every coefficient and the intercept is positive. RD-GA appears to find only the former configuration in all runs. Because the GA has identical constant parameters across all runs, it is much more likely for each run to follow a similar evolutionary path as compared to SAGA with its dynamic parameters that are initialized randomly and may evolve differently in each run. Therefore, the evolved individuals are more likely to be more consistent in the GA than in the SAGA, leading to the GA being unable to discover both of the viable coefficient configurations. In terms of the accuracy of the SAGA and GA for this particular problem, this consistency is irrelevant, as both configurations find equivalent results. On other problems, the increased variation across the runs of SAGA may lead to improved results if the GA gets stuck in a local optima in every run.

Fig. 9 shows the plots of the evolved coefficients on the 50:50 data set using only the insignificant variables, defined as every variable except four, six, and nine. The plots of the insignificant variables are much more varied across runs, even on the standardized data methods. A few mostly consistent trends are found, the most prominent being a highly positive coefficient for variable 3, "Number of HCPCS/CPT codes billed", but this is not as impactful on accuracy as the significant variables are. With only the insignificant variables, neither the SAGA nor GA methods are able to find a clear strategy for obtaining accurate results.

8. Conclusions and future work

In this study, we propose a self-adaptive GA and apply it to the problem of classifying Medicare standardized payments as being above or below the industry median. The motivation for developing SAGA is twofold; to improve usability by reducing the amount of parameter tuning necessary and to improve the results by allowing the algorithm to itself determine the most effective parameter settings during the execution of the algorithm. Previous studies on self-adaptation in EAs focus the self-adaptation on only a few parameters in the system. Due to potential non-linear interactions among system parameters, SAGA extends previous approaches by applying self-adaptation to all of the parameters related to the genetic operators.

Self-adaptation is implemented in SAGA by adding a second parameter chromosome to each regular GA solution chromosome. Both chromosomes are subject to the evolutionary process. The second chromosome encodes values for all genetic operators available in a given SAGA run, and the evolved values in any given generation of a run determine the operators that apply in that generation. The operators that may be applied in a given SAGA run are limited to those specified at implementation. The presented implementation includes eight crossover operators and three mutation operators from the GA literature.

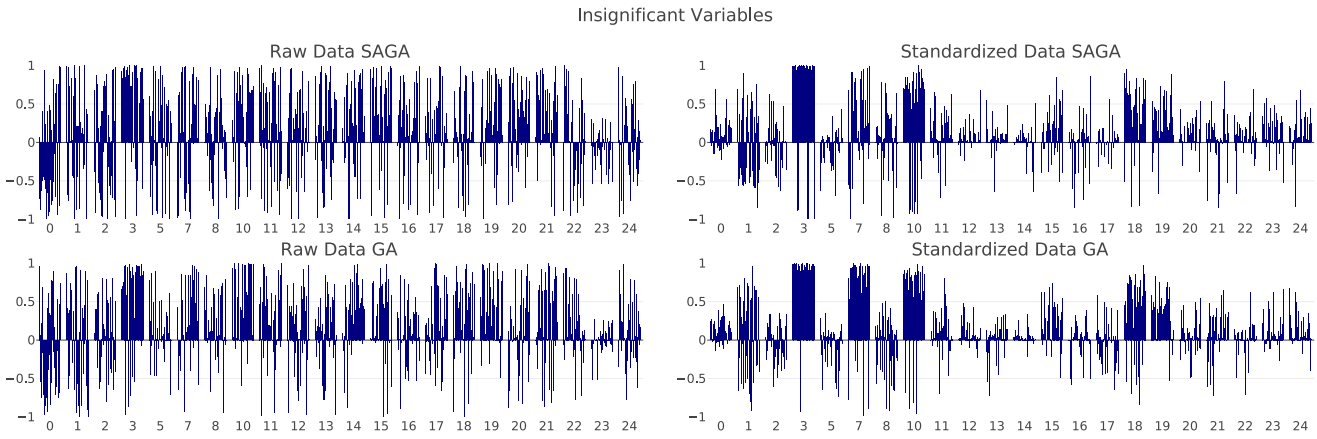

Fig. 9. GAs and SAGA coefficients using only the insignificant variables on the 50:50 training:test set.

We compare SAGA to LR as a benchmark and compare SAGA to a regular GA to ensure that the self-adaptive mechanisms do not adversely affect the algorithm's evolutionary capabilities. The input data consists of 40,662 data points; each containing 25 independent variables which consists of both provider and location information. We divide the input data into training and test sets and run the experiment on five sets of data, each with a different training set/test set ratio and run with both raw and standardized data.

Results show that the SAGA performance of the standardized input data yields the highest accuracy of all methods tested. All but one of the SAGA and GA variants perform equivalent to or superior to LR. SAGA and GA tended to give more positive classification than negative classifications, which is an incorrect ratio since the data set is even split between the two classes. LR on the other hand, found roughly equal number of positive and negative classifications. Despite the positive bias, the SAGA and GA still tended to return a higher accuracy, particularly for the self-adaptive SAGA on standardized data. Across all five of the methods, there is no appreciable degradation of accuracy as the size of the training set decreases until the training size reaches just 1% of the input data.

We run both the SAGA and the GA both with standardized and raw data to see if standardization results in any noticeable differences. Results indicate that standardization produces a small improvement in accuracy for the SAGA and a small decrease in accuracy for the GA. For this particular problem, the nature of the ranges of the independent variables is beneficial to running the algorithms with raw data, since the influence variables are also among the variables who have the largest ranges; Nonetheless, the best results are found with standardized data. Our results suggest that standardization is indeed recommended for SAGAs and GAs on these kind of classification problems, much like other ML methods.

One of the shortcomings of current state-of-the-art methods for predictive modeling, such as LR and deep learning, is that they are unable to provide explanations of how they make classifications. These methods essentially generate a black box that accepts a data point as input and outputs a classification. Analysis of the behaviors of SAGA and GA indicate that they are effectively enacting simultaneous feature selection and classification. This result means that, in addition to learning how to make accurate classifications from training data, our SAGA and GA approach can also provide information on which features are most relevant for making an accurate classification. As a result, in addition to achieving comparable or better classification accuracy as LR, SAGA and GA can also provide users with valuable information about the classification process itself.

With these findings, we propose that a tailor-made SAGA or EA which is purposely designed around simultaneous feature selection and classification can be made to expand EA use on classification problems. One potential fruitful avenue to approach this is through the use of a subtype of EAs that have variable-length chromosomes. In variable length EAs, each individual has a unique length chromosome. For this problem, each individual would include a different subset of the independent variables. Through the evolutionary mechanisms, the EA should then evolve to individuals that contain only the influential subset of independent variables. Self-adaption can then be applied to these variable-length EAs to improve usability.

In conclusion, the primary contributions of this work are as follows:

- We show that SAGA is a competitive method for predictive classification in the field of healthcare informatics.
- We show that, in addition to achieving effective classification, the GA and SAGA methodology can also simultaneously perform feature selection. Feature selection provides valuable domain-specific insights about which are the relevant features that contribute to the classification task.
- Finally, we show that adding self-adaptation mechanisms to a GA successfully eases the burden of turning a GA's system parameters without any detriment to a GA's ability to learn and find a solution. On the contrary, our results indicate that self-adaptation may improve results over a manually tuned GA.

CRediT authorship contribution statement

Reamonn Norat: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft. **Annie S. Wu:** Supervision, Conceptualization, Methodology, Formal analysis, Investigation, Writing – review and editing. **Xinliang Liu:** Conceptualization, Methodology, Resources.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

Acan, A., Altincay, H., Tekol, Y., & Unveren, A. (2003). A genetic algorithm with multiple crossover operators for optimal frequency assignment problem. In *Proceedings of the Congress on Evolutionary Computation, vol. 1* (pp. 256–263).

Alsaeedan, W., & Menai, M. E. B. (2015). A self-adaptive genetic algorithm for the word sense disambiguation problem. *Current Approaches in Applied Artificial Intelligence, LNCS 9101*, 580–590.

American Physical Therapy Association (2014). *Guide to physical therapist practice 3.0, Alexandria, VA*. Available at: http://guidetoptpractice.apta.org/ (Accessed 04 December 2014).

Bäck, T. (1992a). The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm. In *Proceedings of the Second International Conference on Parallel Problem Solving from Nature* (pp. 85–94).

Bäck, T. (1992b). Self-adaptation in genetic algorithms. In *Proceedings of the First European Conference on Artificial Life* (pp. 263–271).

Beyer, H.-G. (1996). Toward a theory of evolution strategies: Self-adaptation. *Evolutionary Computation*, *3*(3), 311–347.

Castro, P. A. D., & Camargo, H. A. (2004). Learning and optimization of fuzzy rule base by means of self-adaptive genetic algorithm. In *Proceedings of the IEEE International Conference on Fuzzy Systems* (pp. 1037–1042).

Cavill, R., Smith, S., & Tyrrell, A. (2005). Multi-chromosomal genetic programming. In *Proceedings of the Seventh Annual Conference on Genetic and Evolutionary Computation* (pp. 1753–1759).

Chaurasia, V., & Pal, S. (2014). Data mining techniques: to predict and resolve breast cancer survivability. *International Journal of Computer Science and Mobile Computing*, *3*(1), 10–22.

Chiba, Z., Abghour, N., Moussaid, K., El Omri, A., & Rida, M. (2019). A clever approach to develop an efficient deep neural network based IDS for cloud environments using a self-adaptive genetic algorithm. In *Proceedings of the International Conference on Advanced Communication Technologies and Networking* (pp. 1–9). IEEE.

Contreras, R. C., Junior, O. M., & Viana, M. S. (2020). A new local search adaptive genetic algorithm for the pseudo-coloring problem. In *Proceedings of the International Conference on Swarm Intelligence* (pp. 349–361).

Contreras-Bolton, C., & Parada, V. (2015). Automatic combination of operators in a genetic algorithm to solve the traveling salesman problem. *PLoS One*, *10*(9), 1–25.

De Jong, K. A. (1975). *An analysis of the behavior of a class of genetic adaptive systems* (Ph.D. thesis), University of Michigan.

De Vasconcelos, M. J. P., Silva, S., Tomé, M., Alvim, M., & Pereira, J. M. C. (2001). Spatial prediction of fire ignition probabilities: comparing logistic regression and neural networks. *Photogrammetric Engineering and Remote Sensing*, *67*(1), 73–81.

Deb, K., & Agrawal, R. B. (1995). Simulated binary crossover for continuous search space. *Complex Systems*, *9*(2), 115–148.

Deb, K., & Beyer, H.-G. (2001). Self-adaptive genetic algorithms with simulated binary crossover. *Evolutionary Computation*, *9*(2), 197–221.

Deb, K., & Deb, D. (2014). Analysing mutation schemes for real-parameter genetic algorithms. *International Journal of Artificial Intelligence and Soft Computing*, *4*(1), 1–28.

Deb, K., Joshi, D., & Anand, A. (2002). Real-coded evolutionary algorithms with parent-centric recombination. In *Proceedings of the Congress on Evolutionary Computation, vol. 1* (pp. 61–66).

Dehuri, S., Patnaik, S., Ghosh, A., & Mall, R. (2008). Application of elitist multi-objective genetic algorithm for classification rule generation. *Applied Soft Computing*, *8*(1), 477–487.

Doerr, B., Witt, C., & Yang, J. (2018). Runtime analysis for self-adaptive mutation rates. In *Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 1475–1482). ACM.

Eiben, A. E., Hinterding, R., & Michalewicz, Z. (1999). Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, *3*(2), 124–141.

Eiben, A. E., & Smit, S. K. (2011a). Evolutionary algorithm parameters and methods to tune them. In *Autonomous Search* (pp. 15–36).

Eiben, A. E., & Smit, S. K. (2011b). Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, *1*(1), 19–31.

Eshelman, L. J., & Schaffer, J. D. (1993). Real-coded genetic algorithms and interval-schemata. In *Foundations of Genetic Algorithms, vol. 2* (pp. 187–202).

Fernández, A., García, S., Luengo, J., Bernadó-Mansilla, E., & Herrera, F. (2010). Genetics-based machine learning for rule induction: state of the art, taxonomy, and comparative study. *IEEE Transactions on Evolutionary Computation*, *14*(6), 913–941.

Fidelis, M. V., Lopes, H. S., & Freitas, A. A. (2000). Discovering comprehensible classification rules with a genetic algorithm. In *Proceedings of the Congress on Evolutionary Computation, vol. 1* (pp. 805–810).

Fogarty, T. C. (1989). Varying the probability of mutation in the genetic algorithm. In *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 104–109).

Galaviz, J., & Xuri, A. (1996). A self-adaptive genetic algorithm for function optimization. In *Proceedings Mexico-USA Collaboration in Intelligent Systems Technologies* (pp. 156–161). IEEE.

Greenwood, G. W., & Zhu, Q. J. (2001). Convergence in evolutionary programs with self-adaptation. *Evolutionary Computation*, *9*(2), 147–157.

Grefenstette, J. J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, *16*(1), 122–128.

Guvenir, H. A., & Erel, E. (1998). Multicriteria inventory classification using a genetic algorithm. *European Journal of Operational Research*, *105*(1), 29–37.

Hadka, D., & Reed, P. (2013). Borg: An auto-adaptive many-objective evolutionary computing framework. *Evolutionary Computation*, *21*(2), 231–259.

Han, S., & Xiao, L. (2022). An improved adaptive genetic algorithm. In *SHS Web of Conferences, vol. 140* (p. 01044).

Hansen, N., & Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, *9*(2), 159–195.

Harik, G. R., & Lobo, F. G. (1999). A parameter-less genetic algorithm. In *Proceedings of the First Annual Conference on Genetic and Evolutionary Computation, vol. 1* (pp. 258–265).

Hartmann, S. (2002). A self-adapting genetic algorithms for project scheduling under resource constraints. *Naval Research Logistics*, *49*, 433–448.

Hesser, J., & Männer, R. (1990). Towards an optimal mutation probability for genetic algorithms. In *Proceedings of the International Conference on Parallel Problem Solving from Nature* (pp. 23–32).

Hinterding, R. (1997). Self-adaptation using multi-chromosomes. In *Proceedings of the IEEE International Conference on Evolutionary Computation* (pp. 87–91). IEEE.

Hinterding, R., Michalewicz, Z., & Eiben, A. E. (1997). Adaptation in evolutionary computation: A survey. In *Proceedings of the International Conference on Evolutionary Computation* (pp. 65–69). IEEE.

Höglund, H. (2017). Tax payment default prediction using genetic algorithm-based variable selection. *Expert Systems with Applications*, *88*(1), 368–375.

Holland, J. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

Hong, T.-P., Wang, H.-S., Lin, W.-Y., & Lee, W.-Y. (2002). Evolution of appropriate crossover and mutation operators in a genetic process. *Applied Intelligence*, *16*(1), 7–17.

Hosmer Jr, D. W., & Lemeshow, S. (1989). *Applied Logistic Regression*. John Wiley and Sons.

Jefferson, M. F., Pendleton, N., Lucas, S. B., & Horan, M. A. (2000). Comparison of a genetic algorithm neural network with logistic regression for predicting outcome after surgery for patients with nonsmall cell lung carcinoma. *Cancer: Interdisciplinary International Journal of the American Cancer Society*, *79*(7), 1338–1342.

Karafotias, G., Eiben, A. E., & Hoogendoorn, M. (2014). Generic parameter control with reinforcement learning. In *Proceedings of the Annual Conference on Genetic and Evolutionary Computation* (pp. 1319–1326).

Karafotias, G., Hoogendoorn, M., & Eiben, A. E. (2015). Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation*, *19*(2), 167–187.

Kita, H., Ono, I., & Kobayashi, S. (2000). Multi-parental extension of the unimodal normal distribution crossover for real-coded genetic algorithms. *Transactions of the Society of Instrument and Control Engineers*, *36*(10), 875–883.

Kivijärvi, J., Fränti, P., & Nevalainen, O. (2003). Self-adaptive genetic algorithm for clustering. *Journal of Heuristics*, *9*(2), 113–129.

Kovačič, M., & Dolenc, F. (2016). Prediction of the natural gas consumption in chemical processing facilities with genetic programming. *Genetic Programming and Evolvable Machines*, *17*(3), 231–249.

Kühn, M., Severin, T., & Salzwedel, H. (2013). Variable mutation rate at genetic algorithms: introduction of chromosome fitness in connection with multi-chromosome representation. *International Journal of Computer Applications*, *72*(17), 31–38.

LaPorte, G. J., Branke, J., & Chen, C.-H. (2015). Adaptive parent population sizing in evolution strategies. *Evolutionary Computation*, *23*(3), 397–420.

Li, Z., Guan, A., Ge, H., & Lian, F. (2017). Wavelength selection of amino acid THz absorption spectra for quantitative analysis by a self-adaptive genetic algorithm and comparison with mwPLS. *Microchemical Journal*, *132*, 185–189.

Lin, W.-Y., Lee, W.-Y., & Hong, T.-P. (2003). Adapting crossover and mutation rates in genetic algorithms. *Journal of Information Science and Engineering*, *19*(5), 889–903.

Lu, H., Wen, X., Lan, L., An, Y., & Li, X. (2015). A self-adaptive genetic algorithm to estimate JA model parameters considering minor loops. *Journal of Magnetism and Magnetic Materials*, *374*, 502–507.

Mambrini, A., & Sudholt, D. (2015). Design and analysis of schemes for adapting migration intervals in parallel evolutionary algorithms. *Evolutionary Computation*, *23*(4), 559–582.

Mather, M., Jacobsen, L. A., & Pollard, K. M. (2015). Aging in the United States. *Population Bulletin*, *70*(2).

Michalewicz, Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer.

Mills, K. L., Filliben, J. J., & Haines, A. L. (2015). Determining relative importance and effective settings for genetic algorithm control parameters. *Evolutionary Computation*, *23*(2), 309–342.

Min, S.-H., Lee, J., & Han, I. (2006). Hybrid genetic algorithms and support vector machines for bankruptcy prediction. *Expert Systems with Applications*, *31*(3), 652–660.

Murata, T., & Ishibuchi, H. (1996). Positive and negative combination effects of crossover and mutation operators in sequencing problems. In *Proceedings of the IEEE International Conference on Evolutionary Computation* (pp. 170–175).

Paul, D., Su, R., Romain, M., Sébastien, V., Pierre, V., & Isabelle, G. (2017). Feature selection for outcome prediction in oesophageal cancer using genetic algorithm and random forest classifier. *Computerized Medical Imaging and Graphics*, *60*, 42–49.

Qin, A. K., & Suganthan, P. N. (2005). Self-adaptive differential evolution algorithm for numerical optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation, vol. 2* (pp. 1785–1791).

Riedel, J., Blum, S., Puisa, R., & Wintermantel, M. (2005). Adaptive mutation strategies for evolutionary algorithms. In *2nd Weimar Optimization and Stochastic Days*.

Sabar, N. R., Bhaskar, A., Chung, E., Turky, A., & Song, A. (2019). A self-adaptive evolutionary algorithm for dynamic vehicle routing problems with traffic congestion. *Swarm and Evolutionary Computation*, *44*, 1018–1027.

Schaffer, J. D., Caruana, R., Eshelman, L. J., & Das, R. (1989). A study of control parameters affecting online performance of genetic algorithms for function optimization. In *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 51–60).

Serban, R., Kupraszewicz, A., & Hu, G. (2011). Predicting the characteristics of people living in the south USA using logistic regression and decision tree. In *IEEE International Conference on Industrial Informatics* (pp. 688–693).

Serpell, M., & Smith, J. E. (2010). Self-adaptation of mutation operator and probability for permutation representations in genetic algorithms. *Evolutionary Computation*, *18*(3), 491–514.

Shrivastava, P., Shukla, A., Vepakomma, P., Bhansali, N., & Verma, K. (2017). A survey of nature-inspired algorithms for feature selection to identify Parkinson's disease. *Computer Methods and Programs in Biomedicine*, *139*, 171–179.

Smith, J., & Fogarty, T. C. (1996). Self adaptation of mutation rates in a steady state genetic algorithm. In *Proceedings of the IEEE International Conference on Evolutionary Computation* (pp. 318–323).

Spears, W. M. (1995). Adapting crossover in evolutionary algorithms. In *Proceedings of the Fourth Annual Conference on Evolutionary Programming* (pp. 367–384).

Srinivasa, K. G., Venugopal, K. R., & Patnaik, L. M. (2007). A self-adaptive migration model genetic algorithm for data mining applications. *Information Sciences*, *177*, 4295–4313.

Sun, N., & Lu, Y. (2019). A self-adaptive genetic algorithm with improved mutation mode based on measurement of population diversity. *Neural Computing and Applications*, *31*(5), 1435–1443.

Thierens, D. (2002). Adaptive mutation rate control schemes in genetic algorithms. In *Proceedings of the Congress on Evolutionary Computation, vol. 1* (pp. 980–985).

Thornblade, L. W., Flum, D. R., & Flaxman, A. D. (2018). Predicting future elective colon resection for diverticulitis using patterns of health care utilization. *Journal for Electronic Health Data and Methods*, *6*(1), 1–8.

Tsutsui, S., Yamamura, M., & Higuchi, T. (1999). Multi-parent recombination with simplex crossover in real coded genetic algorithms. In *Proceedings of the First Annual Conference on Genetic and Evolutionary Computation, vol. 1* (pp. 657–664).

Vandewater, L., Brusic, V., Wilson, W., Macaulay, L., & Zhang, P. (2015). An adaptive genetic algorithm for selection of blood-based biomarkers for prediction of Alzheimer's disease progression. *BMC Bioinformatics*, *16*(18), S1.

Venugopal, K. R., Srinivasa, K. G., & Patnaik, L. M. (2009). Self adaptive genetic algorithms. In *Soft Computing for Data Mining Applications, vol. 190* (pp. 19–50).

Wright, A. H. (1991). Genetic algorithms for real parameter optimization. In *Foundations of Genetic Algorithms, vol. 1* (pp. 205–218).

Wu, A. S., Liu, X., & Norat, R. (2019). A genetic algorithm approach to predictive modeling of Medicare payments to physical therapists. In *Proceedings of the Florida Artificial Intelligence Research Society Conference* (pp. 311–316).

Xie, H., & Zhang, M. (2013). Parent selection pressure auto-tuning for tournament selection in genetic programming. *IEEE Transactions on Evolutionary Computation*, *17*(1), 1–19.

Xue, Y., Xue, B., & Zhang, M. J. (2019). Self-adaptive particle swarm optimization for large-scale feature selection in classification. *ACM Transactions on Knowledge Discovery from Data*, *13*(5), 1–27.

Yang, X., Li, Y., Liu, F., Lan, T., Teng, L., & Sarkar, T. K. (2021). Antenna position optimization method based on adaptive genetic algorithm with self-supervised differential operator for distributed coherent aperture radar. *IET Radar, Sonar & Navigation*, *15*(7), 677–685.

Yoon, H.-S., & Moon, B.-R. (2002). An empirical study on the synergy of multiple crossover operators. *IEEE Transactions on Evolutionary Computation*, *6*(2), 212–223.

Yu, H., Yao, Z., Sui, X., Gu, G., & Chen, Q. (2022). Focusing through disturbed multimode optical fiber based on self-adaptive genetic algorithm. *Optik*, *261*, Article 169129.

Yuan, Y., & Wang, G. (2019). Self-adaptive genetic algorithm for bucket wheel reclaimer real-parameter optimization. *IEEE Access*, *7*, 47762–47768.

Zhang, Z., Trevino, V., Hoseini, S. S., Belciug, S., Boopathi, A. M., Zhang, P., Gorunescu, F., Subha, V., & Dai, S. (2018). Variable selection in logistic regression model with genetic algorithm. *Annals of Translational Medicine*, *6*(3), 45.

Zhou, W., Qin, S., & Zhou, C. (2022). AGV path planning based on improved adaptive genetic algorithm. In *Proceedings of the International Conference on Artificial Intelligence and Computer Information Technology* (pp. 1–4). IEEE.