GENETIC ALGORITHM REPRESENTATION SELECTION IMPACT ON BINARY CLASSIFICATION PROBLEMS

By
STEPHEN MALDONADO

A thesis submitted in partial fulfillment of the requirements
for the Honors in the Major Program in Computer Science
in the College of Engineering and Computer Science
and in the Burnett Honors College
at the University of Central Florida
Orlando, Florida

Summer Term 2022

ABSTRACT

In this thesis, we explore the impact of problem representation on the ability for the genetic algorithms (GA) to evolve a binary prediction model to predict whether a physical therapist is paid above or below the median amount from Medicare. We explore three different problem representations, the vector GA (VGA), the binary GA (BGA), and the proportional GA (PGA). We find that all three representations can produce models with high accuracy and low loss that are better than Scikit-Learn's logistic regression model and that all three representations select the same features; however, the PGA representation tends to create lower weights than the VGA and BGA. We also find that mutation rate creates more of a difference in accuracy when comparing the individual with the best fitness (lowest binary cross entropy loss) and the most accurate solution when the mutation rate is higher. We then explore potential of biases in the PGA mapping functions that may encourage the lower values. We find that the PGA has biases on the values they can encode depending on the mapping function; however, since we do not find a bias towards lower values for all tested mapping functions, it is more likely that it is more difficult for the PGA to encode more extreme values given crossover tends to have an averaging effect on the PGA chromosome.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

We investigate the performance of genetic algorithms with different problem representations in optimizing linear prediction models on the Medicare Physical Therapist dataset. Prediction modeling consists of finding relationships between variables in a set of known data and allows us to predict unknown values given known data. Genetic algorithms (GA) can be used to optimize linear prediction models by evaluating and evolving individuals that encode a linear prediction model as chromosomes, but there are many different methods of encoding information in a genetic algorithm. The different methods of encoding information in genetic algorithms are known as problem representation. Different problem representations encode information differently, which can evolve different solutions. Using a data set containing information about physical therapist offices, we will demonstrate how well genetic algorithms can optimize linear models in a binary classification task through logistic regression to predict whether the physical therapy offices are paid above or below median Medicare payment. We then compare the effects that problem representation has on the binary prediction models created.

In the 21st century, we are producing more data than able to be analyzed by human labor. With advancements in artificial intelligence (AI) and machine learning (ML), we can use machines to analyze the data and find relationships between variables through predictive modeling. These models have been used in industries like insurance, health care, meteorology, finance, and many other areas. Research into predictive modeling allows us to improve these models or better train these models to have higher accuracy, and in turn allow us to make better predictions and better decisions.

Problem representations provide different means of encoding solutions in chromosomes. Since different problem representations encode information differently, each representation creates a different search space leading to potentially different performances on the same problems. We will implement several representations and compare the binary cross entropy loss, accuracy, and the weights to see how different representations perform on optimizing linear prediction models. Comparing the binary cross entropy loss, accuracy, and weights with different representations will allow us to compare how these representations act on the same problem. Research into problem representation allows future researchers to better understand which representation may be most effective for their problem.

We will compare the models created by the GAs with different problem representations and parameters and optimize logistic regression models on the Medicare Physical Therapist dataset. We will compare the weights extracted and the loss and accuracy scores of the evolved models and see how they differentiate between the different representations.

PREDICTIVE MODELING

Predictive modeling (Gkisser, 1993) is a popular problem in the AI and ML communities. There are many types of prediction models. Different predictive models can be implemented in different instances depending on what is being predicted, the details of the data set, and whether knowing the relationship between the features and the labels is important. These models need to be trained using ML optimization algorithms or evolutionary computation (EC) algorithms.

The predictive modeling problem has a lot of terms specific to the problem. *Training data* refers to the data we use to train or fit the prediction models using an optimization algorithm. We will refer to each data point in a data set as an *instance*. The known values of instances are *features*, and the values we are trying to predict for an instance are called *labels*. Algorithms learn the relationships between the features and the labels of each instance in the training data to predict the labels when provided the features of unlabeled data. *Test data* refers to labelled instances not included in the training data that can be used to validate the predictive model performance on unseen data. Running the prediction model on the test data and comparing it to the true labels is called *validation*. Validating on a test set allows us to see if we are over-fitting. *Over-fitting* occurs when the relationships the model extracts from the training data is too specific to the training data and not generalized.

Prediction models are relationships between features and labels defined in a way that they can take features as input and output a label. Prediction models can take different forms, like equations or sets of rules. The model itself needs to be trained with an optimization algorithm to develop these equations or rules to apply to the features to get the labels as outcome. Some of

these models are simple, and easily interpreted by humans, whereas others have relationships that are too complex for easy interpretation.

Logistic Regression

Logistic regression (Tolles & Meurer, 2016) is used to create prediction models that produce a binary prediction, whether something is True or False. An example can be whether an item belongs in a certain category or exceeds a certain threshold, meaning that the labels are discrete values, 0 or 1, which makes it different from linear regression.

For the following math notation, we will represent the number of instances with N_i and the number of features with N_F. We will represent the features with F_n and the corresponding weights with W_n. The variable C will represent a constant. We will represent residuals for instances with R_i. The predicted label for an instance will be represented with Y_i and the true label of an instance will be represented with L_i.

Logistic regression operates using an optimization algorithm to optimize the bias and a set of weights to apply to the features of instances in a data set. The sigmoid function is applied to the Logistic regression's predictions. The sigmoid function places the predictions on a "S" shaped curve. The purpose of the "S" shaped curve is to push values towards either the extremes, true (1) or false (0).

Logistic regression produces a model that estimates the likelihood that a set an instance belongs to a label. If the predicted likelihood is above 50% (or 0.50), the predicted label is true, otherwise it is labeled false. We calculate prediction using Equation *1* and will represent the predicted likelihood for an instance with P_i.

$$P_i = C + \sum_{n=1}^{N_F} (F_{i,n} * W_n) \quad (1)$$

When creating a logistic regression model, we cannot use the sum of the residuals squared to optimize the model like linear regression since the predicted labels and ground truth labels are discrete values. Instead, logistic regression maximizes the maximum likelihood.

Maximum likelihood is the sum of the predicted odds of each instance being in its correct category as shown in Equation *2*.

$$Fitness = \sum_{i=1}^{N_i} \begin{cases} P_i, & L = True \\ 1 - P_i, & L = False \end{cases} \quad (2)$$

Using the predicted chance when the label is true and 1 - predicted chance when the label is false, we always add the predicted chance that it predicts the correct label. For example, if the model predicts a 0.10 that an instance's label is true and the instance's label is true, we will add 0.10. If the model predicts 0.10 that an instance's label is true, but the instance's label is false, we will add 0.90.

$$Fitness = -\sum_{i=1}^{N_i} [(y_i * \log(P_i) + (1 - y_i) * \log(1 - P_i)] \quad (3)$$

Equation *3* shows how we can better write the maximum likelihood function using y_i, where y_i is 1 when the instance's ground truth label is true and 0 when its ground truth label is false. By making it negative, we now have what is called Binary Cross Entropy loss. We can now use the loss function in our GA as a minimization problem, where we adjust the weights and the constant with the objective of reducing the loss.

Non-Linear Predictive Modeling

This thesis focuses on training logistic regression models, but other forms of predictive models exist. Regression trees and classification trees create a tree of decisions to predict or classify labels for instances. K-Nearest Neighbor classification and regression classifies instances based on the instances most like itself. Neural networks algorithms can also be used but are much more complicated and the relationships are not as easily interpreted.

Regression and classification trees (Breiman, Friedman, Olshen, & Stone, 1984) can be used in place of linear and logistic regression. These models, instead of creating a set of weights, create a set of questions in the form of a tree where the final nodes are the predictions. These models create a step-by-step decision making process that can be interpreted by humans (Wu, et al., 2007).

K-Nearest Neighbor (KNN) (Fix & Hodes, 1989; Altman, 1992) classification and regression uses the "K" most similar instances to decide on how to classify an instance. If K is 1, KNN will classify the instance as the same as the closest individual. If K is greater than 1, it will factor in the K closest instances' labels, whether by averaging them or by using the most frequently occurring. KNN does lack the ability to explain the relationships, and predictions are

not directly based off the relationships between the features and the labels, but rather how similar they are to instances trained on known labels.

Neural networks consist of connected layers of nodes where data is passed through each layer and sets of weights are applied to them. Although they often have high accuracy, the relationships they develop between the features and the labels are not easily interpreted by humans, meaning they may not be ideal for all applications. Understanding the relationship between the features and the predicted labels is important in some applications. In such applications, neural networks may not always be useful. An example could be a bank firm denying a loan. The bank may be required to explain to a consumer why they are refusing to give a loan to an individual, but if they utilize a neural network to make the decision, they won't be able to explain to the customer the reason behind the denial.

Evolutionary Algorithms in Predictive Modeling

Evolutionary computation has been successfully applied to prediction problems in previous works. Fernandez et al (Fernandez, Garcia, Luengo, Bernado-Mansilla, & Herrera, 2010), Dehuri et al (Dehuri, Patnaik, Ghosh, & Mall, 2008), and Fidelis et al (Fidelis, Lopes, & Freitas, 2000) utilize GAs to develop a set of rules to organize data into categories. Kovacic and Dolenc (Kovacic & Dolenc, 2016) use genetic programming to develop an expression for predictions. Previous researchers (Guvenir & Erel, 1998; Norat, 2020; Wu, Liu, & Norat, 2019) create linear models using GAs. Similar to this work, other researchers use GAs as the optimization algorithm for linear regression to create linear models (Ng, Skitmore, & Wong, 2008; Stojanovic, Milivojevic, Ivanovic, Milivojevic, & Divac, 2013; ChatterJee, Laudato, &

Lynch, 1996). These works show that evolutionary algorithms are capable of evolving prediction models to find relationships between the features and the labels (Norat, 2020; Wu, Liu, & Norat, 2019; Desbordes, et al., 2017; Min, Lee, & Han, 2006; Jefferson, Pendleton, Lucas, & Horan, 1997). Some researchers use these relationships to help improve the performance of other ML algorithms (Desbordes, et al., 2017; Jefferson, Pendleton, Lucas, & Horan, 1997; Min, Lee, & Han, 2006).

PROBLEM

We hope to find a binary predictive model to predict whether a physical therapy office will receive a payment from Medicare above or below the median Medicare payment given features about the physical therapy provider and information about the county the physical therapist office is located in. These features come from the 2014 Medicare Provider Utilization and Payment Data: Physician and Other Supplier Public Use File (PUF) and the 2015-2016 Health Resource File (AHRF).

This data set contains 30,498 physical therapist offices. We split this data set into two data sets, a training set consisting of approximately 66% of the data, and a test set where each set contains approximately 34% of the data. The data consists of both categorical and numerical values. We turn categorical features into separate binary features to represent each possible state as a unique feature. We also standardize the data, as previous work with this data has shown that standardization was more effective than normalizing the data or not applying any normalization or standardization at all (Wu, Liu, & Norat, 2019; Norat, 2020). Table 1 lists all the features from the Medicare physical therapist dataset following preprocessing along with the type of variable. These types include Binary, or True and False values, Float values which are values including decimals, and Integer values which are whole numbers.

Table 1

List of features for the Medicare Physical Therapist data set.

Feature	Type
Large Metro Area	Binary
Medium Metro Area	Binary
Non-Metro or Missing Area	Binary
Small Metro Area	Binary
Avg. Age of Beneficiaries	Float
Avg HCC Risk Score of Beneficiaries	Float
Female Physical Therapist	Binary
Male Physical Therapist	Binary
Median Household Income (2014)	Float
Medicare FFS Beneficiary Avg HCC Score (2014)	Float
Medicare FFS Beneficiary Avg Age Fee for Service (2014)	Float
Number of HCPCS/CPT Codes Billed	Integer
Number of Medicare Beneficiaries	Integer
Standardized Risk Adjusted Per Capita Medicare Costs	Float
Charge Allowed Amount Ratio	Float
No Doctorate in Physical Therapy	Binary
Doctorate in Physical Therapy	Binary
Standardized Medicare Payment per Beneficiary	Float
Primary Care Physicians per 10K population (2014)	Float
% 65 or Older in Deep Poverty (2014)	Float
% Medicare Beneficiary Eligible for Medicare	Float
% Medicare FFS Beneficiaries Female (2014)	Float
Medicare FF Beneficiaries	Float
Proportion of Physical Agent	Float
Proxy for Number of New Patients	Float
Physical Therapist Beneficiary Ratio	Float
Number of Physical Therapists per 10K (2009)	Integer
% of Therapeutic Procedures	Float

We can investigate the features to get a better understanding of the dataset to understand which features are expected to be emphasized in our models. Figure 1 demonstrates the mean values and their respective 95% confidence interval sorted by the label, where each subfigure is a feature of the dataset and the mean of physical therapists' features below the median Medicare payment amount are on the left side of the subfigures and the mean of the physical therapists' features above the median Medicare payment amount are on the right side of the subfigures.

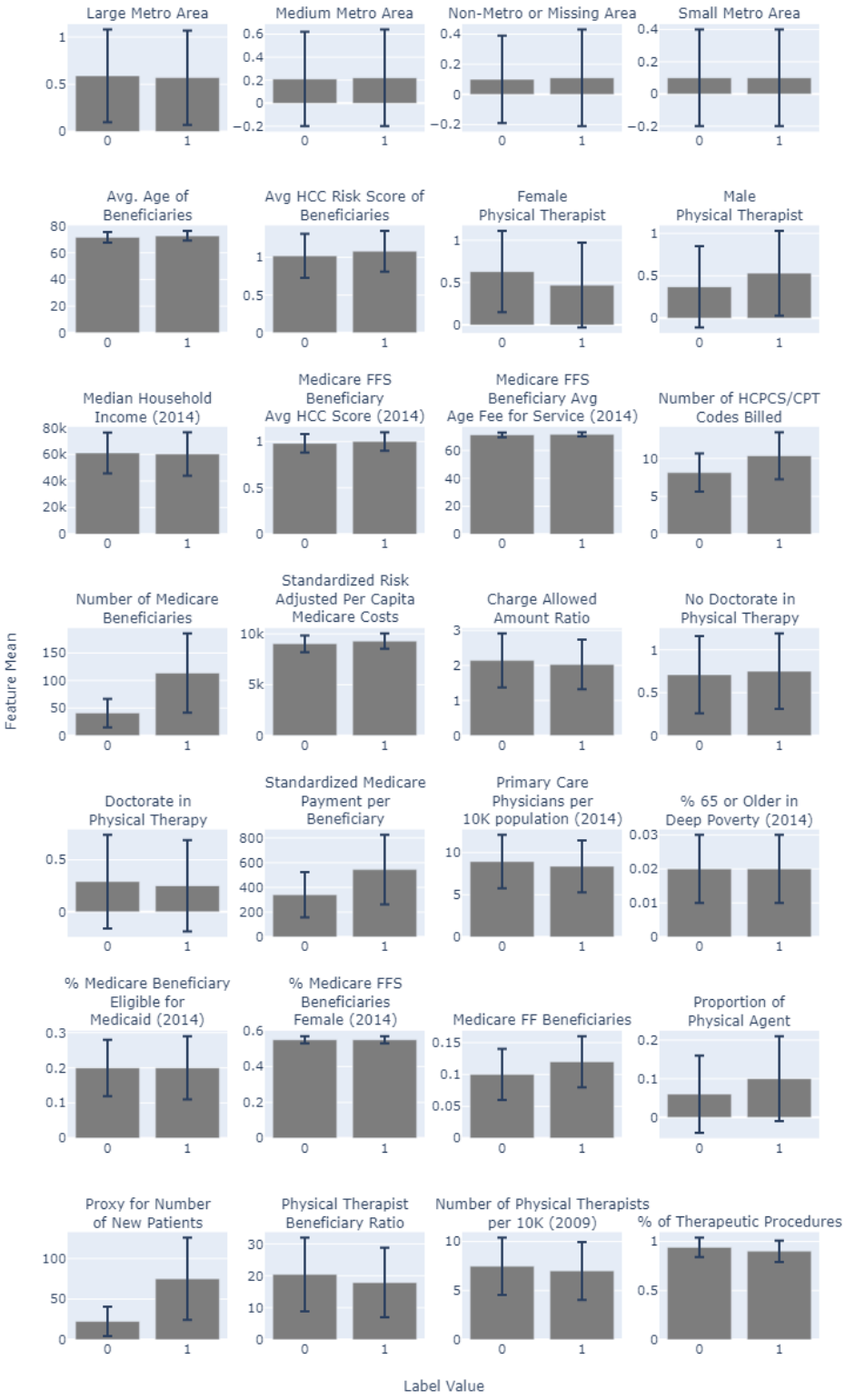

Figure 1: Table demonstrating the means of the features of the Medicare Physical Therapist Dataset Organized by Label

Figure 1 demonstrates a larger difference in the mean for several features, with the most significant differences being “Number of Medicare Beneficiaries”, “Standardized Medicare Payment Per Beneficiary”, and “Proxy for Number of New Patients”. We can expect that these features are likely to have higher weights as there is a larger difference between the features’ means when sorted by label.

Figure 2 demonstrates the Pearson correlation between the features including the label. The values represented by yellow demonstrate the highest positive correlation and the values represented by dark blue represent the highest negative correlation. The leftmost column of Figure 2 demonstrates that “Number of Medicare Beneficiaries”, “Standardized Medicare Payment Per Beneficiary”, and “Proxy for Number of New Patients” features have the highest correlations with the “Above Median Payment” label. The higher correlations with the labels are further evidence that we are likely to have higher weights associated with these features.

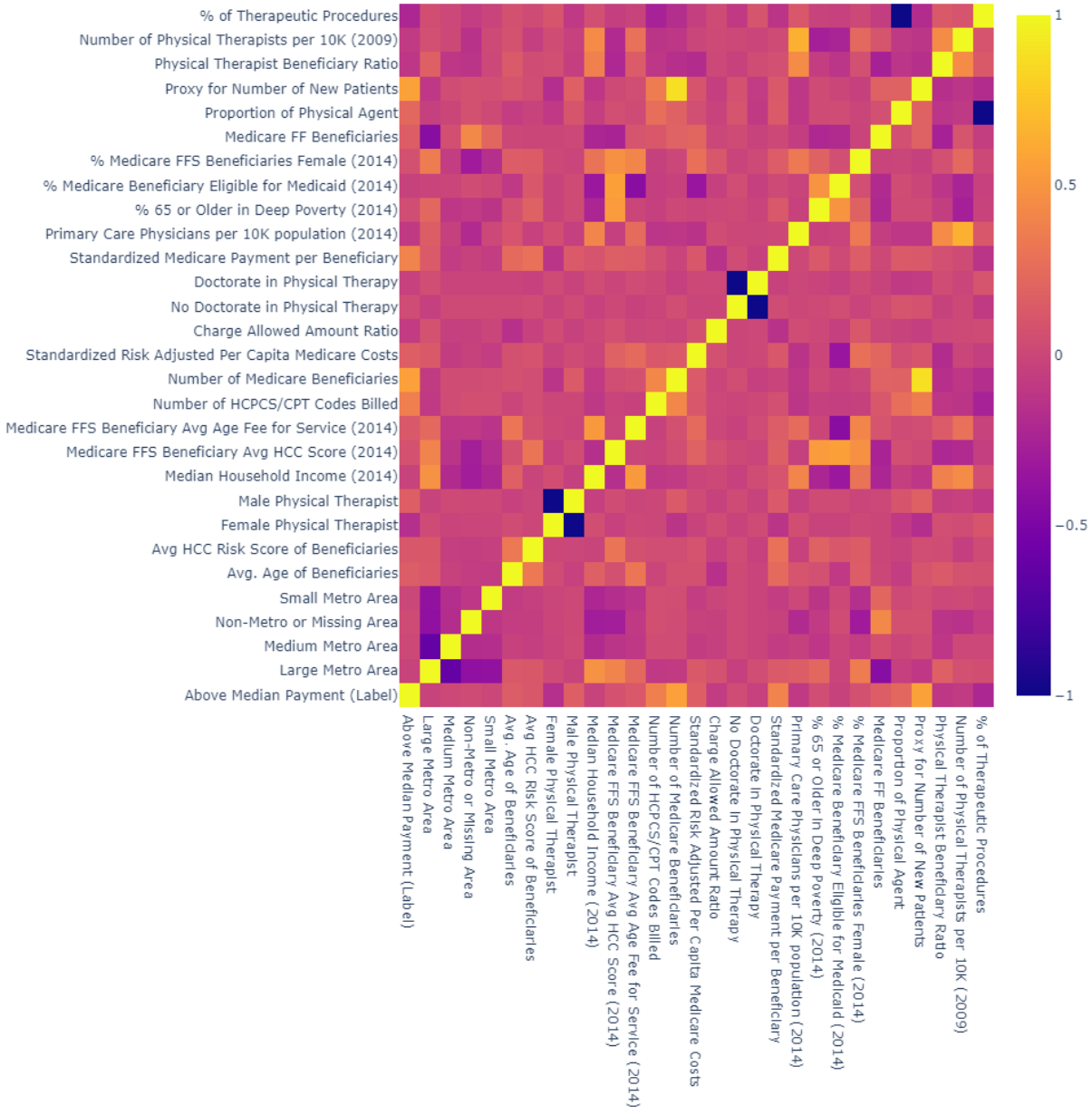

Figure 2: Correlation between the features and the label

GENETIC ALGORITHMS

The Genetic Algorithm (GA) was developed by John Holland in the 1970s and is an optimization algorithm that simulates biological principles like DNA, natural selection, and reproduction to evolve optimal solutions to problems. The possible solutions the GA can search in is referred to as the *search space*. The GA starts with a *population* of candidate solutions called *individuals*. How the individual represents the solution is known as the *problem representation*, and it can impact the performance and behavior of a GA. The quality of an individual is measured by its *fitness*, which is calculated using a *fitness function*. Then, a *selection method* is used to select individuals, usually prioritizing more fit individuals, to become parents for the next generation, simulating natural selection. *Crossover*, operators that combine parents' solutions, and *mutation*, operators that randomly modify the children, are used to create children from the parents and simulate how crossover and mutation may occur in biological systems. This cycle repeats, until hitting some implemented stopping criteria, like a maximum number of generations. *Algorithm 1* demonstrates the steps the generational genetic algorithm takes to evolve solutions.

Algorithm 1: Generational Genetic Algorithm

```
1: procedure GA RUN(total # of gens)
2:     Initialize the population with random solutions
3:     Evaluate the initial population, assigning each individual a fitness score
4:     current generation # ← 0
5:     while current generation # < total # of gens do
6:         Select parents from the population based off fitness
7:         Perform Crossover to create children
8:         population ← children
9:         Mutate population
10:        Evaluate population, assigning each individual a fitness score
11:        current generation # ← current generation # + 1
12:    return best_individual
```

Search Space

The search space of a problem refers to the set of total possible solutions. GAs are often used when the search space is complicated and too large to perform a brute force search. Different representations and fitness functions lead to search spaces with different landscapes and different sizes.

The *landscape* of a search space refers to the quality of the solutions at different points of the search space. The landscapes that are the easiest for optimization algorithms contain a singular point, the global optima, in which all potential solutions leading to that point are increasing in quality. This sort of landscape is not typically the case, as real-world problems may have a landscape with multiple points that are better than its surroundings but are not the global optima. These are referred to as local optima.

The size of a search space refers to the number of solutions it contains. In many cases the search space does not contain all possible solutions, rather all solutions that a representation can encode. Smaller search spaces may be easier to search but increasing the search space may include a better global optimum.

Population and Individuals

In a genetic algorithm, *population* refers to the total set of individuals in the current generation. An *individual* is a chromosome which encodes a candidate solution to the problem and has a fitness score assigned by the fitness function. The solution to the problem is typically encoded in a set of genes inside the chromosome, and how those genes are encoded in the chromosome is dependent on the problem representation.

Problem Representation

Problem representation refers to how the solution is encoded in the individual. Problem representation can affect the search space and have different properties that affect how the GA behaves. Having different search spaces for the same problem and having different properties means that different representations may lead the GA to perform differently on the same problem. The differences between problem representations stresses the importance of research into representations, as different representations may be better in different instances. We will go over some problem representation options later.

Location Dependent Representations and Positional Bias

Location dependent representations refer to problem representations where the location at which something is encoded determines what it is encoded, meaning certain pieces of information about the solution can be found at fixed locations in the chromosomes. Location dependent representations are often used in GAs as they are easier to implement, can typically be implemented using a shorter chromosome than location independent representations, and are often easier to be interpreted by humans.

Although location dependent representations have their perks, they do introduce something called *positional bias* (Eshelman, Caruana, & Schaffer, 1989). Positional bias is the bias that genes near each other are more likely to end up in the same child following crossover as compared to genes that are further apart. For example, if we had a chromosome with four genes labelled A, B, C, and D and sorted in that order, A and B are more likely to end up with the same child than A and D. Positional bias is not inherently bad. If two related genes are closer together then they may benefit from positional bias; however, GAs are often used on problems that are

poorly understood so it may not be possible to place related values near one another. Positional bias can influence the ability of GAs to develop solutions by disrupting building blocks being created if two related genes are further apart and more frequently separated.

Fitness Function

The fitness function evaluates the solutions encoded in the individuals and assigns them fitness scores depending on the quality of the solution. The fitness function is what determines the optimal solution and the landscape of the search space.

Selection Method

The selection method is the procedure followed to select parents to create the next generation. Most selection methods prioritize selecting individuals with higher fitness scores to become the next generation's parents. They often incorporate probabilistic behavior to allow less-fit individuals to occasionally have the chance to enter the pool of parents to preserve diversity and encourage exploration.

Mutation and Crossover

Crossover and mutation are genetic operators that use the selected parents to create new individuals to continue exploring the search space. These operators are one of the most important aspects of GAs, as they allow the creation of new solutions that are similar to the preexisting solutions that were selected to be the best of the population but are different enough to be able to evaluate a new area of the search space.

Crossover

The crossover operator combines two parent individuals to create two new child individuals[1]. The children created are a mixture of the two parents. Different crossover methods have different approaches to mixing them. Some distribute pieces or segments of the parent's chromosomes among the children randomly, others could produce an average or a weighted average of its genes' values and pass that to their children. Crossover operators can be implemented in various ways depending on what is needed by the problem and what makes sense for the problem representation. Higher crossover rates mean there is a higher chance that two parents will create children using crossover. The alternative to crossover is to remain in the population unchanged.

Mutation

The mutation operator acts on the children following crossover to create random changes to encourage more exploration. Mutation prevents premature convergence to a local optima instead of the global optima, meaning it helps prevent the GA's population from crowding in a good solution rather than finding the best solution. Mutation is how we can introduce new genetic material into the population. Higher mutation rates lead to children who are more different from their parents. If the mutation rate is too high, we may lose the good parts of the solution that the parents were encoding.

[1] Many crossover operators require two parents and create two children; however, the amount of parents required and amount of children that are created vary among different crossover operators.

PROBLEM REPRESENTATION

Choosing the problem representation for a particular application can be difficult, as implementing multiple representations take a long time to implement and evaluate whether it is an effective representation for that application. No ideal representation exists for all problems (Wolpert & Macready, 1997). Each representation has unique properties that can alter the search space and the GA's performance. We examine several different problem representations and explain how their *chromosome values*, or values that are inside the chromosome, are mapped to *encoded values*, the values represented by the chromosome. We also dive into the characteristics of the representations along with properties that make them different from one another.

Table 2 demonstrates different characteristics of the problem representations used in this thesis, including the vector representation, the binary representation, proportional representation.

Table 2

Comparison of Representation Characteristics

	Vector	Binary	Proportional
Location Independent			X
Length Impacts Precision		X	X
Non-Coding Regions	X[2]	X[2]	X

[2] Noncoding regions for the vector and binary GAs are possible by inserting chromosome values that are not used in the decoding process; however, that is not evaluated in our experiments

Location independent means that a representation does not place their genes' values at fixed locations in the chromosome. A representation that is location independent will not experience positional bias (Eshelman, Caruana, & Schaffer, 1989).

Length impacts precision refers to whether the length of the individual chromosome impacts how precise the solution can become. For representations where length is impactful, chromosome that is too short may not be able to encode the best solutions, but a chromosome that is too long may make the search space too large, thus too difficult for a GA to efficiently traverse the search space.

We discuss for each of the representations we investigate how the chromosome encodes its solution, its location dependency, how the chromosome values are mapped to the encoded values, and any other special properties.

Vector GA (VGA)

The Vector GA (VGA) is a simple and direct representation that uses an array of values where each index in the chromosome directly represents an encoded value. It is location dependent, as the encoded values are fixed to certain positions in the chromosome. An example of a VGA chromosome being mapped to encoded values is demonstrated in *Figure 3*.

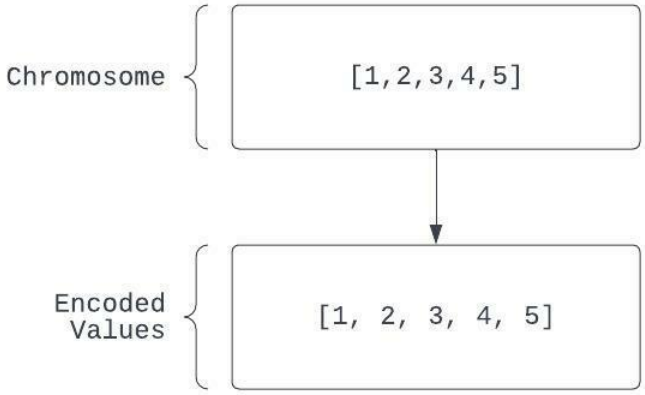

Figure 3: Vector GA Mapping

Problems that require integers or float values do not require a mapping function for the VGA since the chromosome values are the same as the encoded values. Mapping functions like Random Keys can be applied to turn a VGA chromosome into a permutation (Bean, 1994) or repair functions can be used to handle turning invalid solutions that violate constraints into valid solutions (Orvosh & Davis, 1994). Other mapping functions can be applied depending on the needs of the problem.

Binary GA (BGA)

The Binary GA (BGA) is a representation that uses an array of 0s and 1s where subsections of the array represent encoded values. These subsections are fixed regions of the BGA chromosome, making the BGA location dependent. The BGA requires mapping if the problem's inputs are not binary strings. The BGA also has a limited precision based on the length of the chromosome string.

Problems that require a float or integer value will need to be mapped from the binary chromosome values to the encoded values. Figure *4* demonstrates how the BGA representation decodes a chromosome of binary values into encoded values. The process begins with splitting into evenly sized chunks, which will each represent one encoded value. Converting the binary segments into integers using the standard binary integer formatting[3] to produce intermediate encoded values. *Intermediate encoded values* are the values produced between the chromosome values and the final encoded values. We can then map those intermediate encoded values to a range given by a user who provides a minimum and maximum value that are valid for a given problem to produce the final encoded values. To map chromosome values to integer

[3] To convert a binary value to an integer value, one goes from right to left multiplying by 2^{i-1}, where i is the ith value from the right (most right value be multiplied by 2^{1-1})

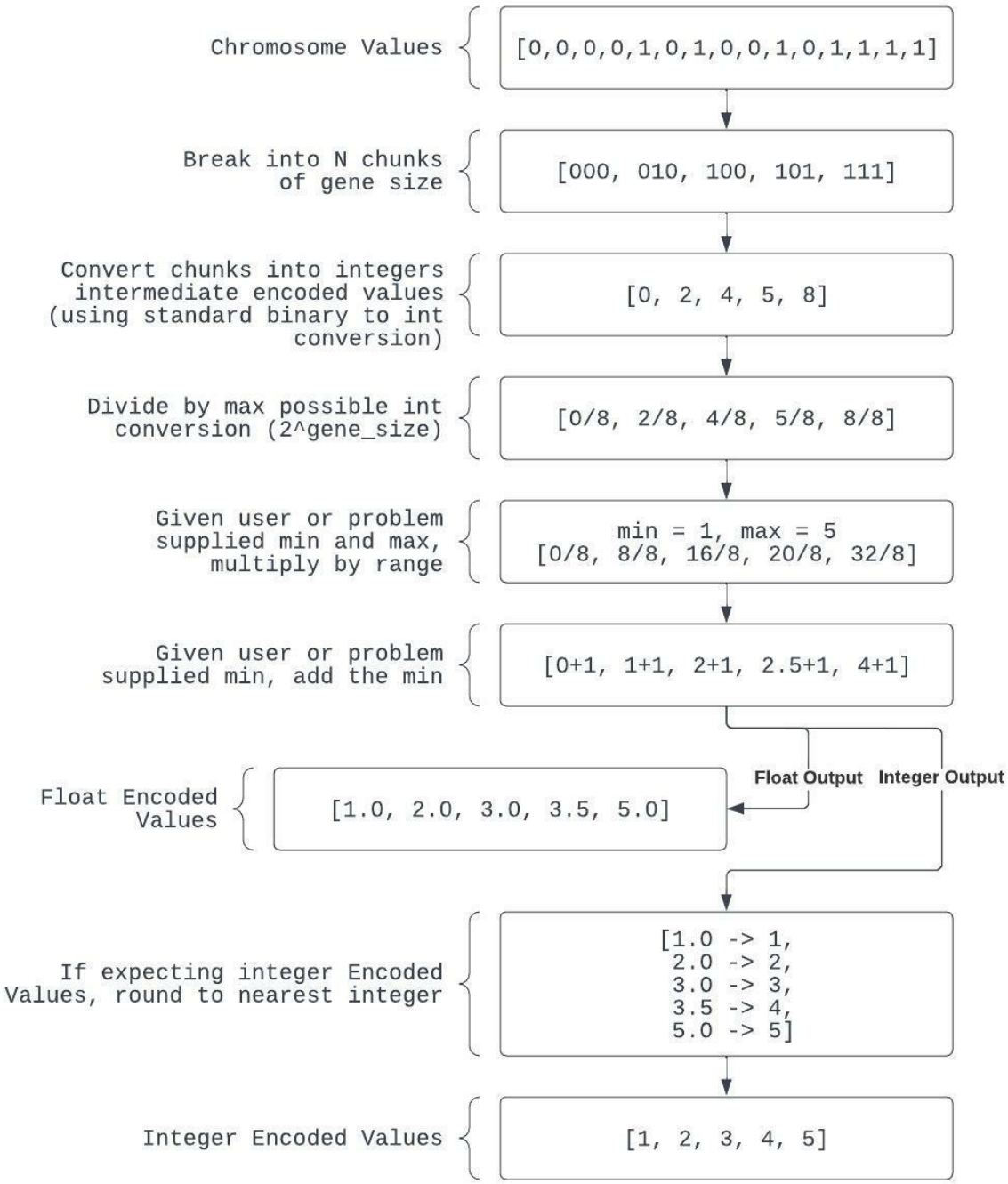

Figure 4: Binary GA Chromosome Mapping

encoded values, the same process can occur but followed by rounding to the nearest integer. For example, in Figure *4*, 0.5 will round to 1 and 3.5 will round to 4.

Longer chromosomes allow more bits to be dedicated to each encoded value, allowing for greater precision. We will call the number of bits dedicated to each encoded value to be the gene size. Each additional bit added to the gene size doubles the number of possible encoded values. With integers, you will need at least $Log_2(R)$ bits to be able to encode every possible integer value, where R is the range of the values you are mapping to. Table 3 shows the number of solutions associated with different gene sizes. Although longer lengths may allow more unique encoded values that could possibly be part of the best solution, it may be harder for the GA to optimize the larger landscape. Shorter lengths on the other hand may create easier landscapes to traverse; however, they may not include the most optimal values necessary for a solution.

Table 3

Binary Gene Size and Number of Solutions

Gene Size	Number of Solutions
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
N	2^N

Proportional GA (PGA)

The Proportional GA (PGA) (Wu & Garibay, 2002) consists of a string of characters where encoded values are represented using an assigned character's count or multiple assigned characters' counts. The PGA representation is location independent since any letter can occur anywhere on the string. The original paper (Wu & Garibay, 2002) introduces several methods

for mapping the chromosome values to encoded values. The PGA also can have a variable length (resolution) chromosome which can change precision (Wu & Garibay, 2004). Homologous Crossover (Burke, De Jong, Grefenstette, Ramsey, & Wu, 1998) can be implemented with this representation to allow variable lengths children in crossover; however, we will not be using that crossover in this paper. The PGA also includes noncoding regions by using noncoding characters.

Figure 5 demonstrates how the proportional representation encodes solutions to problems. Mapping starts with counting each unique character to determine what "proportion" they make up of the chromosome. We apply a mapping function to the count of the characters to calculate intermediate encoded values. Three mapping functions provided by the original authors of the PGA are demonstrated in the diagram (Wu & Garibay, 2002) but any mapping can be applied to the frequencies. Several of the mapping functions from the original paper (Wu & Garibay, 2002) take those frequencies and map them to float values for gene values. For utilization in problems requiring integers, rounding can be used to map the float values to the nearest integer they represent, or one can create an alternative mapping function.

Wu and Garibay (Wu & Garibay, 2002) provides three example mapping functions. Other mapping functions can be applied if another mapping function makes more sense for the problem. We will go over the three equations provided by Wu and Garibay.

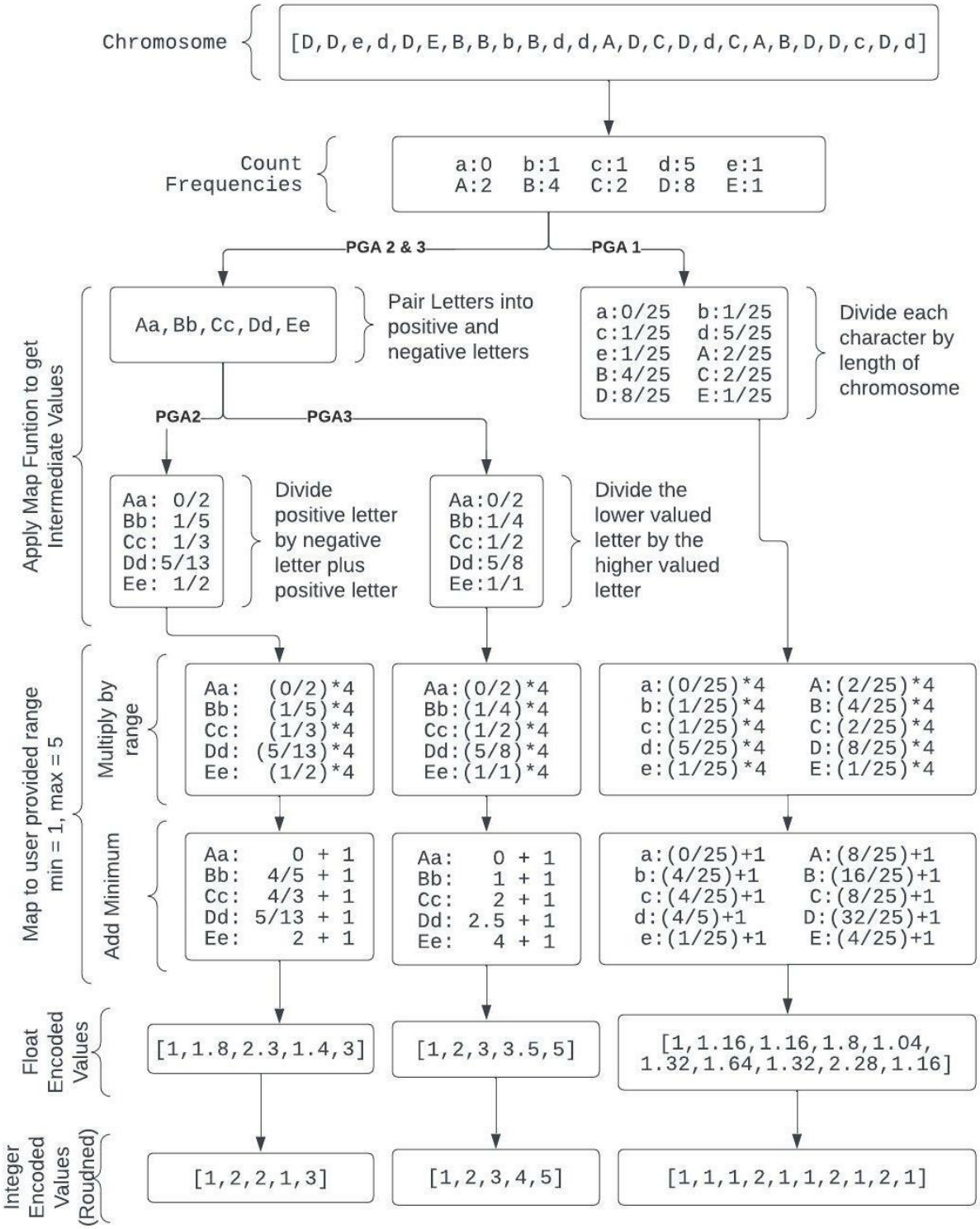

Figure 5: Proportional GA Chromosome Mapping

PGA1 is demonstrated by Equation *4*. PGA1 uses a singular letter per encoded value and creates encoded values between 0 and 1 based off the proportion of the letters in the total chromosome.

$$PGA_1 = \frac{Count\ of\ Letter}{Length\ of\ Chromosme} \tag{4}$$

PGA2 is demonstrated in Equation *5*. PGA2 uses two letters for every gene, where one letter will be called the positive letter, and the other will be called the negative letter. In Figure 5, the pairs are determined by pairing the lowercase letter to its upper-case letter (for example, A and a), where the lowercase letter is the positive letter, and the uppercase letter is the negative letter.

$$PGA_2 = \frac{Count\ of\ Positive\ Letter}{Count\ of Positive\ Letter + Count\ of\ Negative\ Letter} \tag{5}$$

PGA3 is demonstrated in Equation *6*. PGA3 is a simplification of PGA2 that just uses the smaller of the negative and positive character count for the numerator and the larger of the negative and positive character count as the denominator.

$$PGA_3 = \frac{Min(Count\ of\ Positive\ Letter, Count\ of\ Negative\ Letter)}{Max(Count\ of\ Positive\ Letter, Count\ of\ Negative\ Letter)} \tag{6}$$

Like the BGA, the precision of the PGA's encoded values is limited by length of the chromosome. Unlike the VGA and BGA, altering one chromosome value in the PGA can modify up to two encoded values at once as turning one character (for example, A) into another character (for example, B), means that we are changing the count of two characters (A and B). Altering a chromosome value leads to the change of one to two encoded values.

Figure 5 demonstrates how the same PGA chromosome encoding can lead to different kinds of gene values. Different map functions have different advantages and disadvantages depending on their use case. For PGA1, each gene only requires one character and the sum of all the intermediate encoded values will sum up to the value of 1. For PGA2 and PGA3, each gene requires two characters and are used together to form a fraction. Increasing the size of the chromosome allows for more precise values since the numbers in the numerators and the denominators of the map functions' equations can be larger. The original paper (Wu & Garibay, 2002) refers to this concept as "resolution". An implementation of the PGA found that higher resolutions may not work as well on more complicated problems but allow better solutions on simpler ones (Wu & Garibay, 2004). Larger chromosomes allow for more precise values, but also increase the size of the search space. Choosing the proper resolution is important as if it is too small, the PGA may only be able to generate mediocre solutions, and if too large the PGA may not be able to efficiently traverse the landscape.

Variable length chromosomes are possible depending on how crossover is implemented. Variable length PGA implementations have been found to outperform or provide competitive solutions with fixed length PGA implementations (Wu & Garibay, 2002; Wu & Garibay, 2004; Yu & Wu, 2005). Having the ability to evolve a proper resolution may allow solutions to be encoded that are not possible with fixed length chromosomes depending on the chromosome size (Wu & Garibay, 2002).

Homologous crossover (Burke, De Jong, Grefenstette, Ramsey, & Wu, 1998) is possible on the PGA, which selects a window in parent 1 and then searches for the most similar window in parent 2. If the similarity breaches a certain threshold of similarity, there is another

probability of applying crossover. If applying crossover, a random point in that window for parent 1 and the closest match in parent 2 is selected and then used as a split point for crossover. Different split points among the two parents will allow differently sized children.

The PGA can include noncoding regions by having noncoding letters, letters that are not assigned to a particular gene. Including noncoding letters can help create solutions that would require a lower resolution by dedicating the finite space in the chromosome to no genes at all.

METHODOLOGY

In this chapter we discuss the experiment setup. We first lay out how we encode binary classification models. We then go into detail of the tested parameters per representation. Each solution evaluated, we record the binary cross entropy loss as the fitness and record the accuracy of the solutions on both a training data set and a testing data set.

Encoding Binary Classification Models

To encode all values necessary to encode a binary classification model, we need to encode a constant, a weight per feature, and an additional value per feature to determine whether we are going to include a weight for that feature.

We call that additional value per feature that determines whether we are going to include a weight for a feature a *toggle* as it toggles whether a feature will be used in the model. If that toggle gene value is below zero, then the weight gene value is ignored and the value is zeroed or interpreted as zero. If the value is above zero, we utilize the gene value as the weight for the binary prediction model. We can rewrite the predicted likelihood values from Equation *1* into Equation 7, which takes in account the toggle value for a feature n represented by T_n.

$$P_i = C + \sum_{n=1}^{N_F} \begin{cases} 0, & T_n \geq 0 \\ F_{i,n} * W_n, & T_n < 0 \end{cases} \quad (7)$$

We find that introducing the toggles greatly increased the performance of the GA. Other work has introduced a secondary gene that would apply an exponent to the weights when using GAs encoding binary prediction models (Wu, Liu, & Norat, 2019; Norat, 2020). We found using either toggles or exponents greatly increased the ability for the GA to perform; however, we

chose to utilize toggles as introducing toggles decreases the amount of features a model needs to include.

Given that the Medicare Dataset has 28 features, we would have 28 weights, 28 toggles, and a singular constant leading to 57 encoded values. The vector GA will consist of a chromosome length of 57 float values. The binary GA chromosome will consist of 57 chunks of binary values (0 and 1) of various lengths according to the selected gene size. The proportional GA will have 57 unique characters for PGA1 and 114 unique characters, or 57 pairs of characters, for PGA2 and PGA3.

Evaluation

Most of the process of evaluating a logistic regression model is written in the logistic regression section of this thesis. For each example in the training and testing data, we calculate a prediction given its unique set of features using Equation *1*. We then calculate the individual's fitness using those predictions using binary cross entropy loss on the examples from the training data as demonstrated in Equation *3*.

We also calculate the accuracy of the individual by rounding the sigmoid of the predictions to the 0 or 1, the labels values, then seeing what percentage is correct. We calculate accuracy for both the predictions made on the training data and for the testing data to see how often an individual is correct on both the data it is learning on and data it has never seen before.

Genetic Algorithm Parameters

In this section, we cover different parameters utilized for different runs. We have a set of representative-independent parameters that are not unique to each representation including

selection parameters, crossover parameters, mutation rate parameters, and other general run parameters. We then include the parameters evaluated specific to each representation.

We discuss the representation-independent parameters that are consistent to all runs and go into detail about how they work. For selection, we use tournament selection with tournament size of ten and a win chance of 100%. For crossover, we use Two-Point Crossover with a crossover rate of 90%. For each set of parameters and representation, we perform 50 runs evolving 200 generations each with a population size of 200. Most of these parameters come from previous works performed with building a binary prediction model on the same data set (Wu, Liu, & Norat, 2019; Norat, 2020)

Tournament selection is a selection method that takes a random sample of a user determined size from the population and attempts to return the best individual probabilistically given the win chance. If the most fit individual is not selected, we work our way from the most fit to the least fit individual, with a "win chance" percent chance if we should return that individual. We used a tournament size of ten, and we used a 100% win chance meaning the best individual from that sample is always selected.

Two-point crossover is a crossover method that takes two random points on a pair of chromosomes and swaps the areas between those points between the two chromosomes. Figure 6 demonstrates how two random points ("Pt1" and "Pt2") are selected at the same points on both chromosomes. We then take what is between those two points and swap them between the chromosomes. This crossover results in both chromosomes having their original values up to

"Pt1", then having the values from the other chromosome until "Pt2", then back to having their original values from "Pt2" forward.

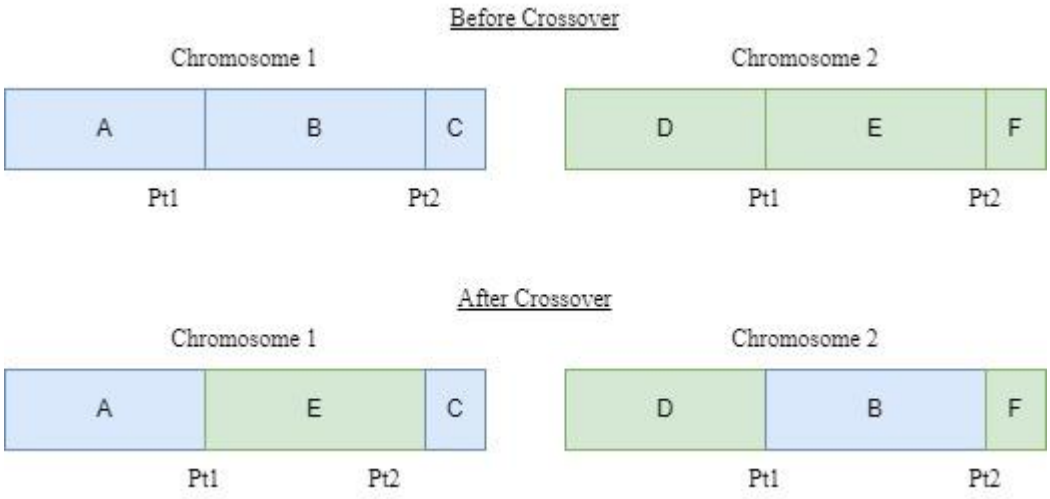

Figure 6: Demonstration of Two-Point Crossover

As for the mutation rates, we run the experiments with the following mutation rates: 0%, 0.1%, 0.5%, 1%, 1.5%, 5%, and 10%. Reviewing different mutation rates allows us to see how it affects the representation.

Lastly, we have 50 runs of 200 generations with a population of 200 individuals per generation. With the start of each run, a new population is created. Given 200 generations, that means there are 200 stages of constructing a new population from the selected parents using crossover and mutation. Each run has 40,000 evaluations[4]. We perform 50 of these runs, meaning we perform in total 2,000,000 evaluations per representation and set of parameters. We record the most fit solution from each of the 50 runs. From these 50 runs' most fit solutions, we

[4] Note 40,000 evaluations does not imply 40,000 unique solutions reviewed, as solutions can be repeated.

record the solution with the best fitness, the solution with the highest accuracy, and the average information of the 50 runs' most fit solutions.

There is only one parameter that is not consistent among the other representations in the VGA, the uniform random mutation. If mutation is applied to a value in the chromosome, it replaces that individual with any other acceptable value between the user inputted minimum value and the user inputted maximum value.

There are two parameters that are not consistent among the other representations in the BGA. These include the mutation operator and the gene size. The BGA in our experiment uses bit-flip Mutation instead of Random Uniform Mutation like the VGA and the PGA. When applying bit-flip Mutation, we "flip the bit", meaning if the value we are mutating is one, we replace it with zero, and if the value we are mutating is zero, we replace it with one. We test various gene sizes to see how they impact the ability to evolve models. We test out 6 bits per encoded value, 12 bits per encoded value, and 18 bits per encoded value. We are hoping to see the effects of different gene sizes to see how it affects the performance and weights generated as a higher gene size allows more precise encoded values to be generated, thus more precise weights.

We review three various parameters with the PGA. We review three mapping functions as laid out in the original paper documenting the PGA. We also test different lengths per character, where the length of the PGA chromosome equals the length per character multiplied by the number of unique characters. We also review how including noncoding characters impact the weights and accuracy of the models. We test three different mapping functions. These three mapping functions are found in the original paper (Wu & Garibay, 2002). These mapping

functions can be found in the Proportional GA (PGA) section and in Equations *4*, *5*, and *6*. Different mapping functions can take the same PGA representation and alter how it behaves. Altering the length per character allows longer or shorter length chromosomes. Increasing or decreasing the length per character alters the resolution, allowing for a different level of precision for the weights. We allow 3, 9, and 15 characters per letter, meaning with three genes and map function one, we would have 9, 27, and 45 unique character long chromosomes. Map function two and three with three genes would have 18, 54, and 90 unique characters long chromosomes as it requires two letters per gene. Table 4 demonstrates how long the chromosomes are depending on the mapping function and the length per character in our experiment.

Table 4

Table indicating length of a PGA chromosome in our experiment depending on the Length Per Character and the Map Function

Length Per Character	Map 1	Map 2	Map 3
3	177	354	354
9	531	1,062	1,062
15	885	1,770	1,770

Lastly, we evaluate how the number of noncoding characters included in the experiment impacts the model's weights and accuracy. We evaluate using 0 noncoding values, 3 noncoding values, and 5 noncoding values. Without including noncoding values, some solutions may not be evolvable (Wu & Garibay, The Proportional Genetic Algorithm: Gene Expression in a Genetic Algorithm, 2002).

RESULTS

In this chapter, we review the binary prediction models evolved by the genetic algorithms. We compare the weights among the representations and view the impact of the parameters per representation on the weights. We then review the differences between the loss and accuracy between representations and the impact of parameters per representation on the loss and accuracy.

Each combination of representations and parameters runs 50 times to form what we call a *run set*. For each run in the run set, we record the solution with the best fitness (lowest fitness). When reviewing the solutions, we look at the most fit, most accurate, and the average solution among all the runs in the run set. The *most fit* solution is the solution with the lowest fitness (lowest binary cross entropy loss) from all 50 runs. The *most accurate* solution is the solution with the highest accuracy from the 50 runs' solutions with the best fitness. The *average solution* consists of the values averaged from all 50 runs' solutions with the best fitness.

Weights

We start by reviewing and comparing the weights of all three representations. Figure 7 demonstrates the weights evolved for the most fit solution (top graph), the most accurate solution (middle graph) and the average solution (bottom graph) for every run set. The x-axis has all the features and the bars above them represent the respective features' weights. We can see that all representations and parameters prioritize the same features, the "Number of Medicare Beneficiaries", "Standardized Medicare Payment Per Beneficiary", and "Proxy for Number of New Patients." These features are the same features we found to have a larger difference in mean and a significant correlation with the label in problem section. We can also note that,

generally, the most fit solutions have slightly lower magnitudes when compared to the most accurate solutions.

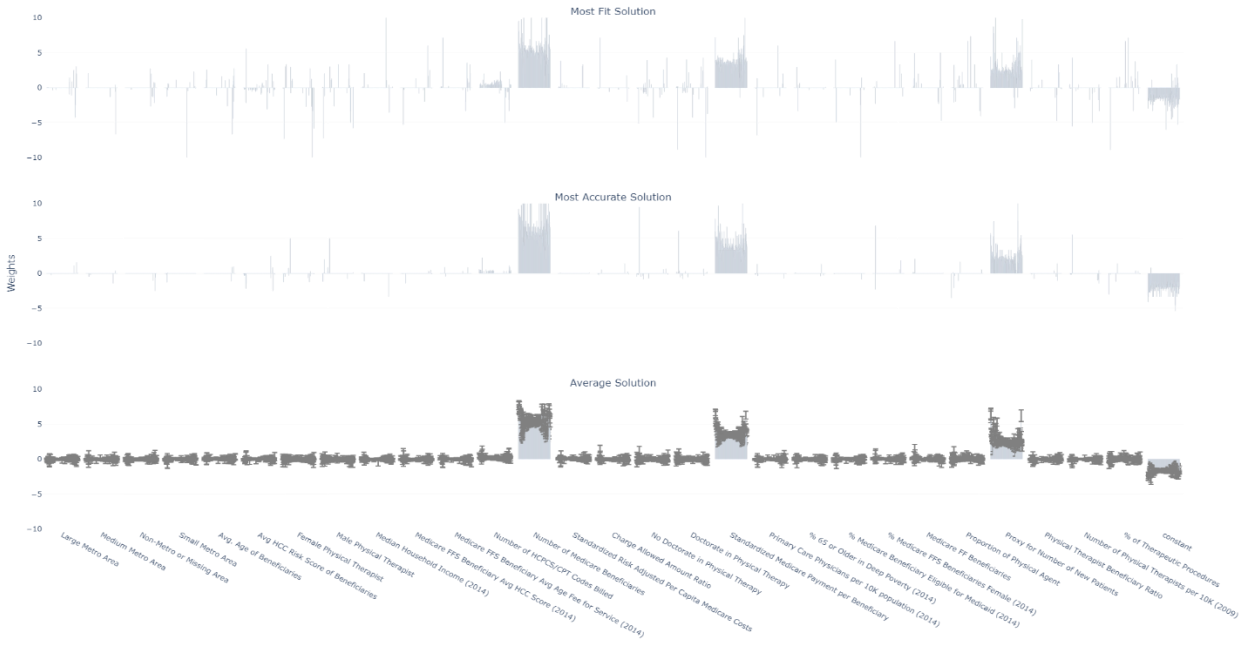

Figure 7: Demonstrates the most fit solutions' weights, the most accurate solutions' weights, and the average solutions' weights for every set of runs

Comparing Weights from Representations

To compare the weights between the different representations, we aggregate the most fit, most accurate, and average solutions' weights per representation regardless of the parameters. Figure 8 is like Figure 7, except Figure 8 demonstrates the average weights for the most fit, most accurate, and average solutions per representation. We can see that BGA (green) and VGA (pink) are more similar than the PGA. PGA1 (Black), PGA2 (Navy), PGA3 (Blue) have notably smaller magnitudes than the other two representations.

Figure 8: Demonstrates the mean weights of the most fit solutions, most accurate solutions, and average solutions per representation aggregating all run sets.

In Figure 9, we demonstrate the weights of the most fit, most accurate, and average solutions from the run set with the parameters that yielded the most fit solution per representation. Reviewing this run set allows us to compare the solutions from the run set with parameters that yielded the solution with the most optimal fitness. Table 5 demonstrates the parameters for each representation that yielded the most fit solution.

Table 5:

Parameters from the run sets with the most fit solution per representation

Representation	Mutation Rate	Gene Size	Length Per Character	Number of Noncoding Characters
BGA	1.0%	12		
PGA1	0.1%		15	1
PGA2	0.1%		15	1
PGA3	0.1%		15	1
VGA	5.0%			

We can see in Figure 9 that the magnitude of the weights from the PGA representation are more similar when looking at the run set that yielded the most fit solutions, particularly when looking at the most fit and average solutions than when we look at the weights aggregated over all run sets regardless of parameters.

Figure 9: Demonstrates the weights of the most fit, most accurate, and average solutions from the run set that contains the most fit solution per representation

In Figure 10 we demonstrate the weights of the most fit, most accurate, and average solutions from the run set that yielded that the most accurate solution per representation. Reviewing this run set allows us to compare the solutions from the run set with the parameters that yielded the solution with the highest accuracy per representation. Table 6 demonstrates the parameters that yielded the most accurate solution per representation.

Table 6:

Parameters from the run set with the most accurate solution per representation

Representation	Mutation Rate	Gene Size	Length Per Character	Number of Noncoding Characters
BGA	5.0%	18		
PGA1	1.5%		9	5
PGA2	1.5%		15	1
PGA3	1.0%		9	5
VGA	5.0%			

Figure 10: Demonstrates the weights of the most fit, most accurate, and average solutions from the run set that contains the most accurate solution per representation

Comparing Effects of Parameters on Weights

We now review the impact of parameters on the weights. We go through each parameter and review the impact of the parameter on the most fit, most accurate, and average weights per representation.

Figure 11 demonstrates the impact of mutation rate on the three weights deemed important. Each row of subfigures still represents the most fit, most accurate, and average solutions. Each column of subfigures represents a different representation. We can see that as mutation rate increases, the confidence interval of the weights increases.

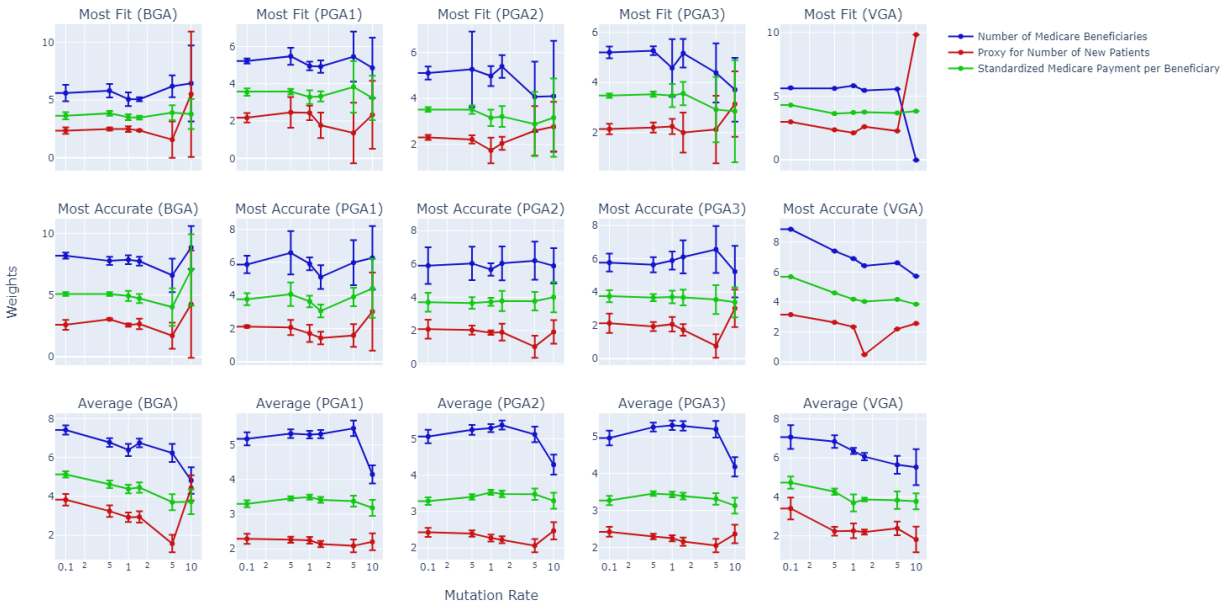

Figure 11: Demonstrates the change of the important weights as mutation rate is increased

Figure 12 demonstrates the three important values' weights change as the gene size is increased for the BGA. We cannot see a significant relationship between gene size and the most fit and most accurate weights.

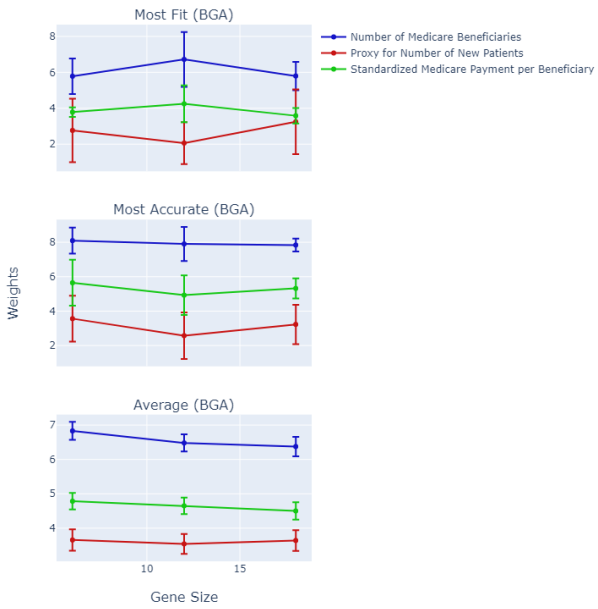

Figure 12: Demonstrates the change of the important weights as gene size is increased

Figure 13 demonstrates the three PGA mapping functions and the impact of length per character and how it affects the weights. We can see that increasing the length per character decreases the weights of the average solutions' weights. We can see that the most accurate solution and the average solution weights decrease as the length per character increases; however, the most fit's trend is different per mapping function.

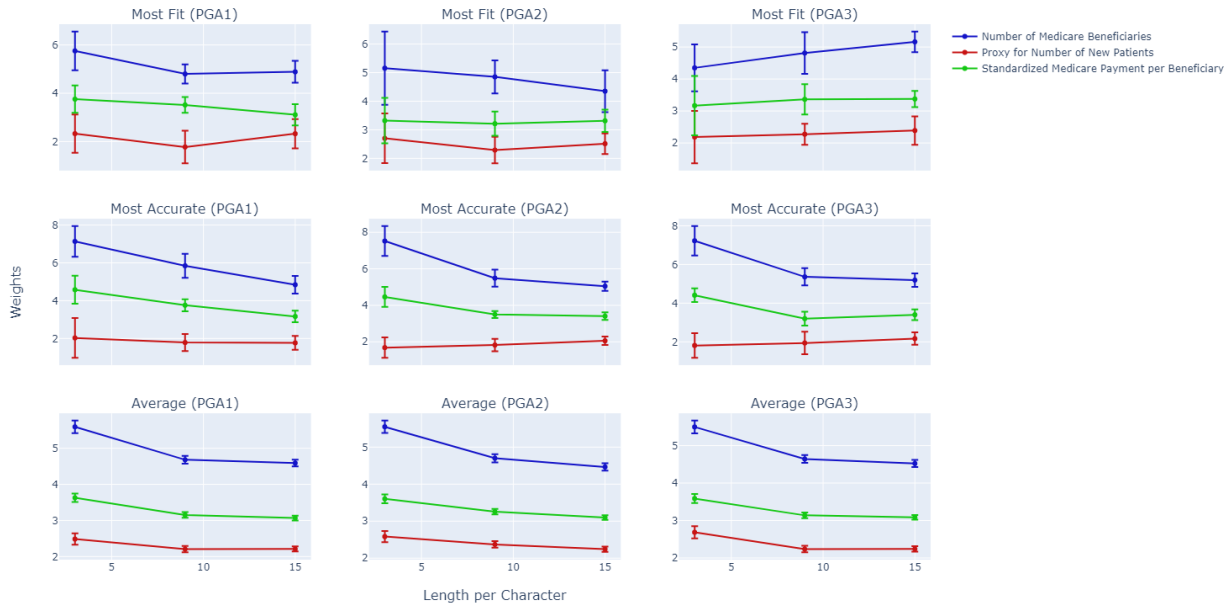

Figure 13: Demonstrates the change of the weights as the Length Per Character increases for the PGA

Figure 14 demonstrates the weights as more noncoding characters are added. It appears the number of noncoding characters does not have an obvious relationship with the evolution of weights. The lack of notable impact of adding noncoding characters could be because any characters associated with unused weights or with toggles can act as noncoding characters. If a feature's weights are disabled, any character associated with that feature are effectively noncoding characters, as modifying the frequency of that character would not affect the solution we are encoding. For characters associated with toggles, if adding or removing those characters does not change whether or not that toggle is enabled or disabled, those characters to an extent also act as noncoding characters. Since the PGA can use characters associated with unused features' weights and can modify characters associated with toggles in moderation without

impacting the solution the PGA chromosome encodes, the PGA can effectively use noncoding character properties without using characters intended to be noncoding characters.

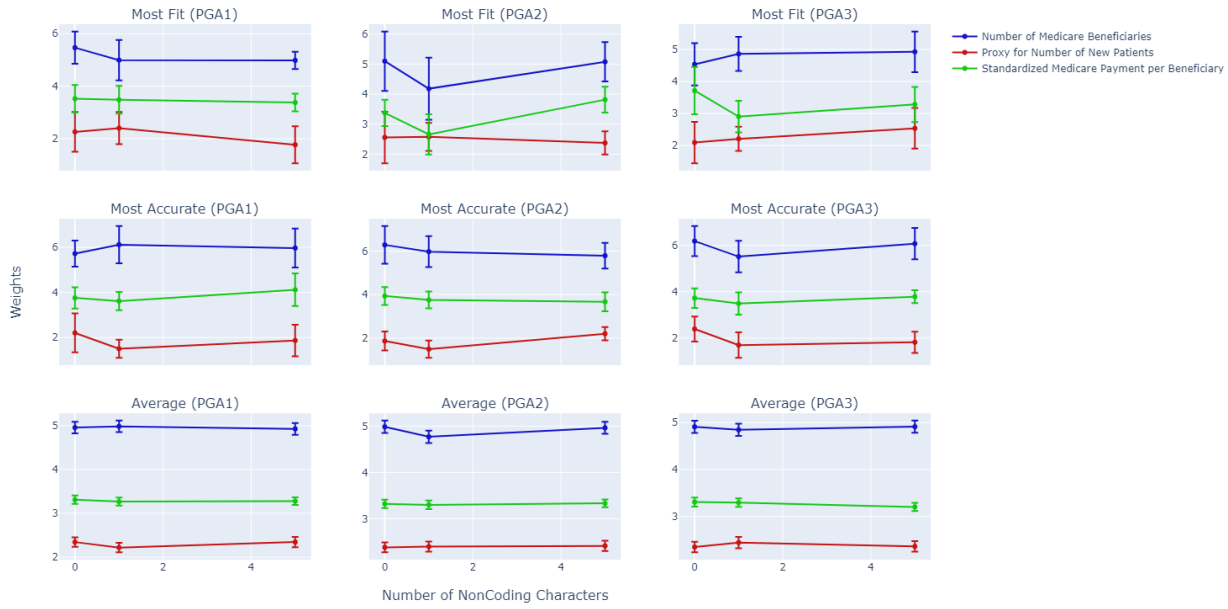

Figure 14: Demonstrates the change of the weights as the Number of Noncoding Character increases for the PGA

Evaluation

We review the evaluation metrics of the binary prediction models we evolve. Evaluation metrics include the binary cross entropy loss (fitness), the training accuracy, and the testing accuracy of the binary prediction models we evolve. Figure 15 demonstrates the relationships between the fitness, training accuracy, and testing accuracy of all the solutions found from every run of every run set. Each point represents the evaluation metrics from a run in a run set. The color demonstrates the representation associated with that run. The top graph demonstrates the relationship between the fitness (x-axis) and the training accuracy (y-axis). Recall that our

fitness is the binary cross entropy loss of the model on the training data, and that our fitness function is a minimization problem. This means that in our case, the most fit solution is the solution with the lowest fitness score. The middle graph demonstrates the relationship between the fitness (x-axis) and the testing accuracy (y-axis). The bottom graph demonstrates the relationship between the testing accuracy (y-axis) and the training accuracy (x-axis).

Figure 15: Demonstrates the relationship between fitness and training accuracy (top), fitness and testing accuracy (middle) and the training accuracy and testing accuracy (bottom)

We can see the relationship between the fitness and training accuracy is like the relationship between the fitness and the testing accuracy. The similarity between these relationships occurs as there appears to be a strong linear relationship between the training and testing accuracy. The linear relationship between training accuracy and testing accuracy demonstrates that the models perform about as well on the training data as they do on the unseen testing data, meaning we are finding weights applicable outside of the training data and likely not overfitting. This linear relationship also shows us that none of these representations tends to overfit on this dataset.

Reviewing the relationship of the fitness and the training accuracy, we can see that the models with the lowest fitness tend to have accuracy ranging from values almost as low as 20% to values as high as the 94.0%; however, higher fitness scores tend to have higher accuracy but occur less frequently. There is not a linear relationship of fitness to training accuracy or fitness to testing accuracy, meaning a model with a good fitness (low loss) value does not imply the model is accurate. A model with low loss and low accuracy tends to make many small mistakes as binary cross entropy would provide a low loss value with very small mistakes, which is a description that would fit many of our runs' solutions. We made a note in the previous section that the most fit solutions tend to have lower weights than the most accurate solutions. Smaller weights could be because we are evolving models with less extreme weights to stay near the decision boundary, minimizing the binary cross entropy function used for fitness.

Comparing Evaluation Metrics from Representations

We review the evaluation metrics aggregated per representation to see how the different representations perform. Figure 16 has several subfigures, demonstrating the fitness (top row), training accuracy (middle row), and testing accuracy (bottom row) of the most fit (left column), most accurate (middle column) and average solutions (right column) aggregated per representation regardless of parameters. Note that the fitness graphs (top row) utilize a logarithmic y-axis to better display all results. We should also note that lower fitness is better fitness since we are using the model's binary cross entropy loss as the fitness.

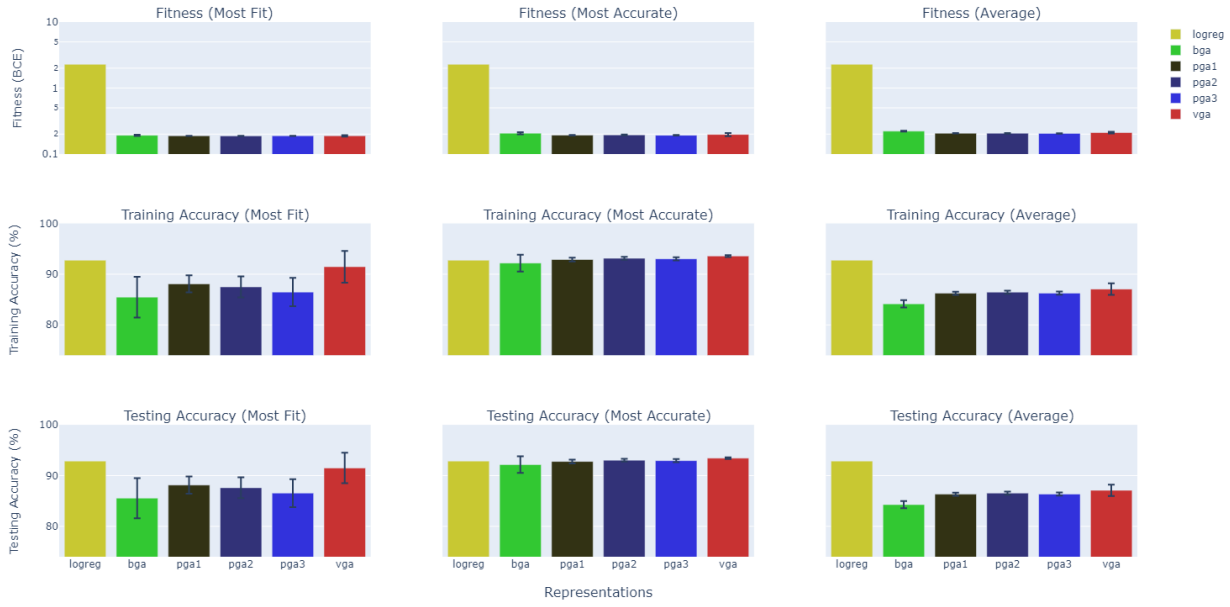

Figure 16: Fitness, Training Accuracy, and Testing Accuracy aggregated by representation for the most fit, most accurate, and average solutions.

In addition to demonstrating each representation, we also include logistic regression model from Scikit-Learn. We use the default parameters from Scikit-Learn except we remove the L2 penalty since the GAs do not use any penalty, and we set the number of iterations to 200

instead of the default of 100 to allow the model to converge. The logistic regression model from Scikit-Learn is labelled as "logreg" in the following figures and is represented by the yellow bars. It should be noted that there is only one run for Scikit-Learn's logistic regression model, meaning that the most fit, most accurate, and average solutions are all the same for Scikit-Learn's logistic regression model.

We can see that all GA representations achieve lower fitness values than Scikit-Learn's logistic regression model. We can also see that, on average, Scikit-Learn's logistic regression model has a higher accuracy than the other representations and has a higher accuracy when compared to the most fit models; however, Scikit-Learn's logistic regression model is on par with the average of the most accurate solutions from all the run sets. Out of the GA representations, the VGA tends to have the highest accuracy on average.

Figure 17 demonstrates the fitness of the most fit, most accurate, and average solutions aggregated by representation regardless of representation zoomed in on the fitness scores produced by the GA. We can see that PGA generally produces solutions with the lowest fitness. The BGA generally has a worse fitness and worse accuracy compared to the other representations. When compared to Scikit-Learn's logistic regression model, the fitness, or binary cross entropy loss, of the Scikit-Learn model is notable higher than any representation produced by the GA.

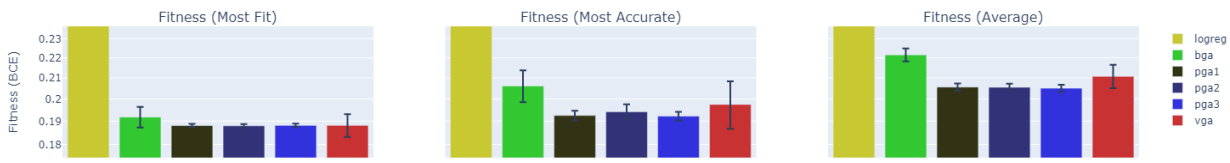

Figure 17: Demonstrates the fitness of most fit, most accurate, and average solutions from all run sets aggregated by representation regardless of parameters, with a reduced y-axis focusing on the GAs

Figure 18 demonstrates the fitness, training accuracy, and testing accuracy of the most fit, most accurate, and average solutions of the run set that contains the solution with the best fitness. Table 5 demonstrates the parameters that are used in the run set with the most fit solution. We can see that most fit solution tends to have higher accuracy than Scikit-Learn's logistic regression model except for PGA2, and that all the most accurate solutions have higher accuracy than Scikit-Learn's logistic regression model; however, on average, the accuracy of the GA tends to be lower than Scikit-Learn's logistic regression model's accuracy when looking at the runset with the most fit solution, especially when looking at the BGA and the VGA. We can also note looking at the most fit run set's average solutions is that the run set that evolves the best solution for the PGA functions also evolves higher accuracy on average with low confidence interval bars when compared to the BGA and VGA.

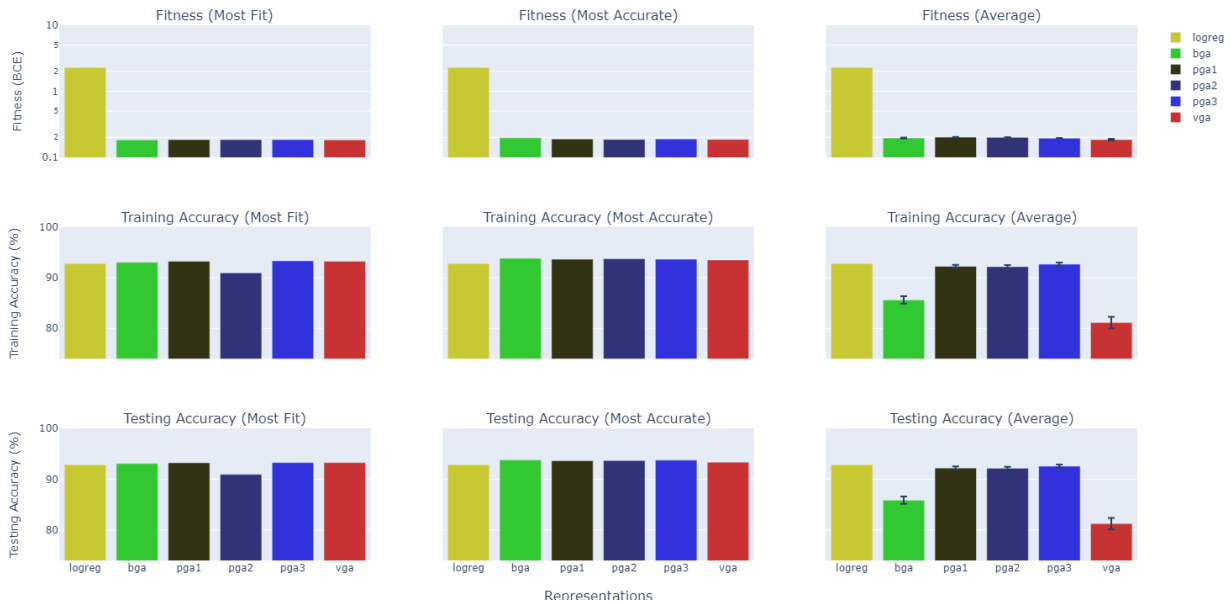

Figure 18: Demonstrates the most fit, most accurate, and average solutions from the run sets that had the most fit solutions per representation

Figure 19 demonstrates the fitness, training accuracy, and testing accuracy of the most fit, most accurate, and the average solutions from the run set that contains the most accurate solution. We should note that the most accurate solution was produced by the BGA and PGA3, both with an accuracy of 94.0%. Despite producing one of the top two most accurate solutions, the BGA has a notably lower training accuracy than other representations.

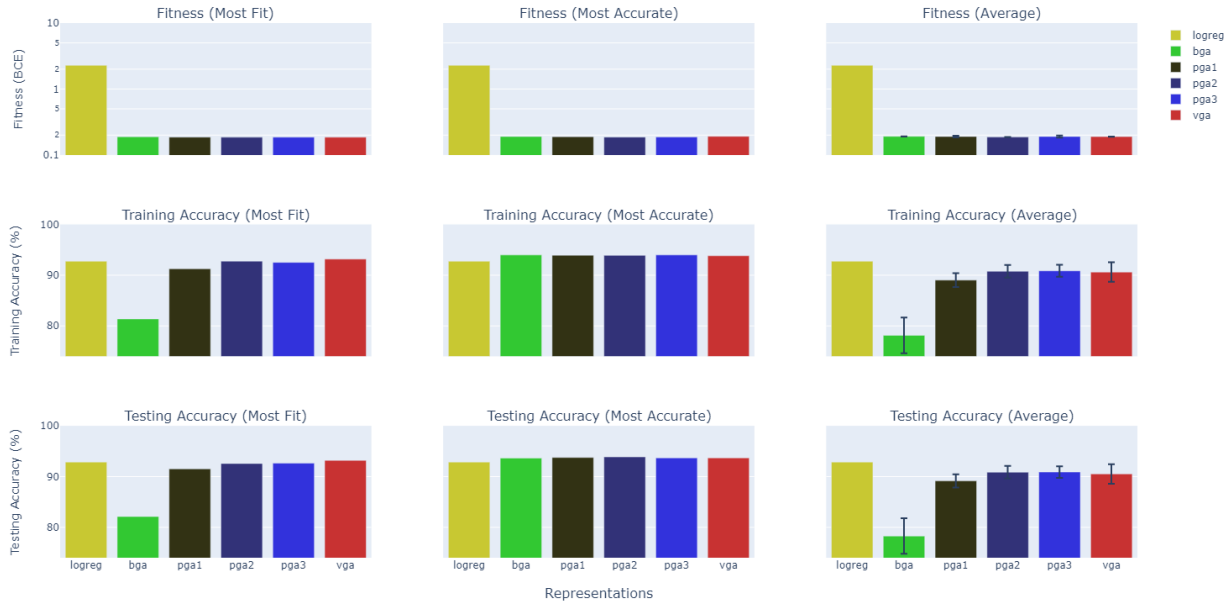

Figure 19: Demonstrates the most fit, most accurate, and average solutions' fitness, training accuracy, and testing accuracy from the run set with the highest accuracy.

Comparing Effects of Parameters on Evaluation Metrics

We now review the impact of different parameters on different the fitness and the training accuracy. Figure 20 demonstrates the impact of Mutation Rate on the fitness (top graph) and the Fitness (top graph) and the Training Accuracy (bottom graph) for the most fit solution (blue),

most accurate solution (red), and the average solution (green). We can see that mutation rate affects the fitness and the accuracy differently. We can note that increasing the mutation rate from 0.1% to 1% although improves the fitness of the average solutions, it also decreases the accuracy of the solutions. Although our goal is to minimize the binary cross entropy loss, we also want to evolve a model with high accuracy. We can see that high mutation rates correlate with reduced accuracy. This correlation means when we are adjusting the GA parameters, particularly mutation rate, we need to review the impact of the parameters on not only the binary cross entropy loss but also the accuracy of the models we are evolving.

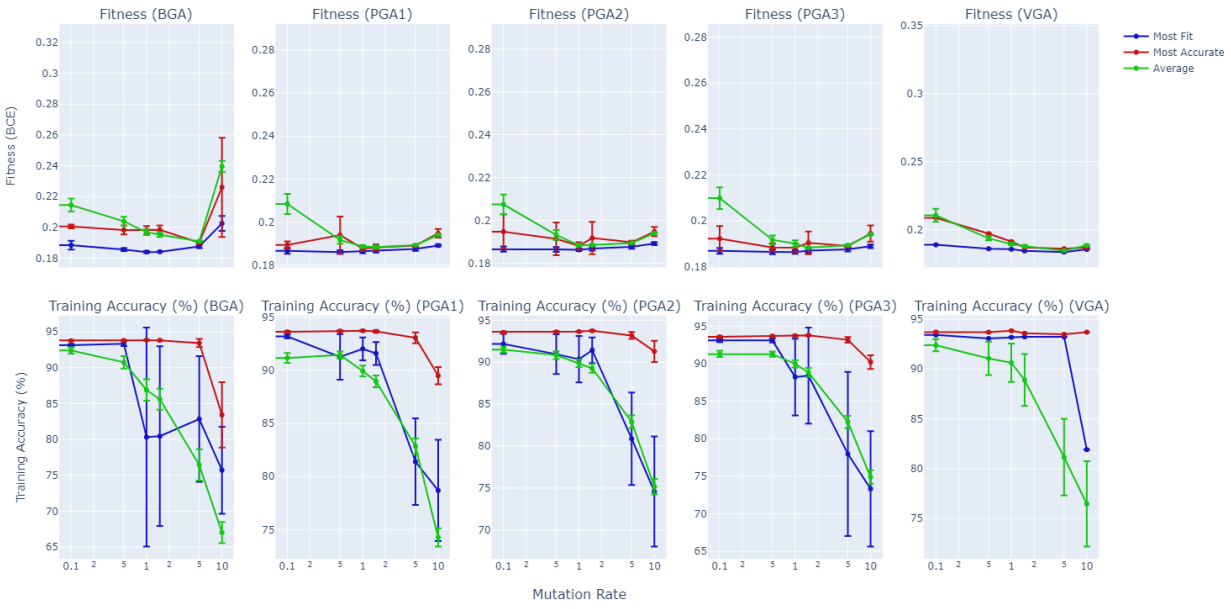

Figure 20: Demonstrates the impact of Mutation Rate on the fitness and training accuracy

Figure 21 demonstrates the impact of gene size on the training accuracy and fitness of the most fit, most accurate, and average solutions. Looking at the average solutions, we can see that

the fitness seems to be lower with a gene size of 12; however, the most accurate solution is most accurate with a gene size of 18 and the most fit solution is most fit with a gene size of 18. This observation demonstrates how increasing the gene size allows more precision for better solutions but also makes the landscape more difficult as, on average, the GA performs worse with higher precision, but a better solution can be found.

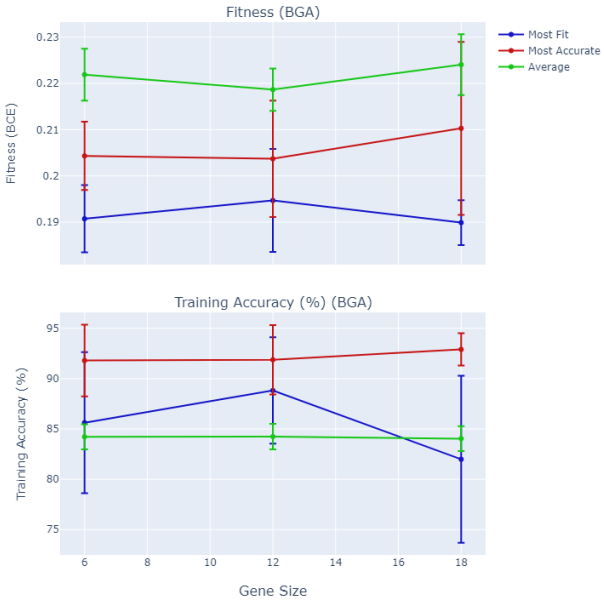

Figure 21 Effect of the Gene Size on the fitness and training accuracy of the BGA

Figure 22 demonstrates the effect of length per character on the fitness and training accuracy of the PGA. Larger length per character implies larger lengths and higher precision. We find that out of the length per characters we tested (3, 9, 15) that the increasing the length per character either lowers the fitness and either increases or does not impact the training accuracy of the most fit, most accurate, and average solutions. This observation does not mean that

increasing the length per character will indefinitely improve or not affect the performance; however, out of the values we tested the higher the length of chromosome generally the better.

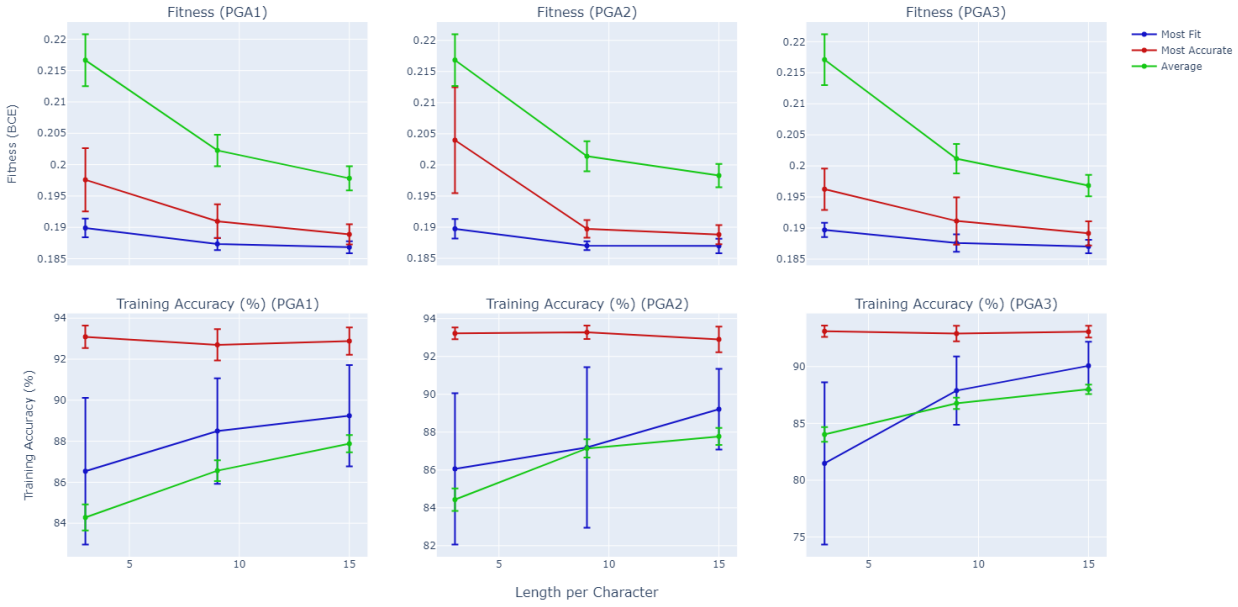

Figure 22 Effect of Length Per Character on the fitness and training accuracy of the PGA

Figure 23 demonstrates the impact of noncoding characters on the fitness and training accuracy of the most fit, most accurate, and average solutions. We can see with the large confidence intervals that there is no notable impact of including noncoding characters on the fitness and training accuracy. This observation follows our discussion from earlier that finds the number of noncoding characters has minimal impact on the PGA on this problem. This is likely due to the PGA's use of characters associated with unused features' weights disabled due to the a

low toggle value or modifying toggle values in such a way that does not breach the threshold to change whether a feature's toggle is enabled or disabled.

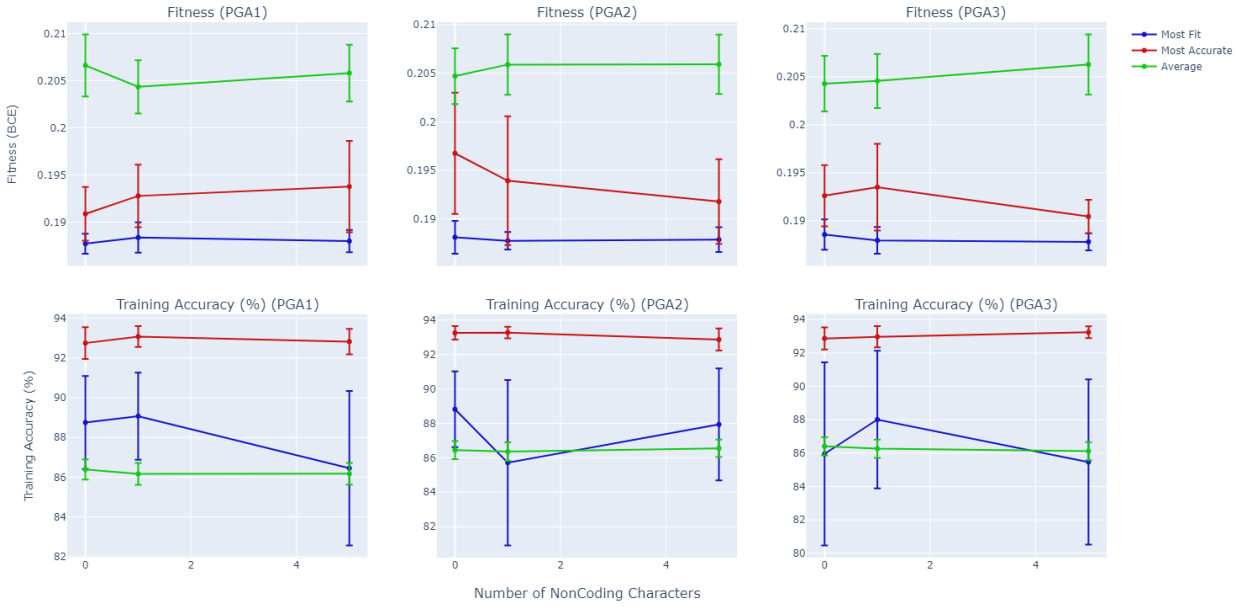

Figure 23 Effect of Number of Noncoding characters on fitness and training accuracy for the PGA

DISCUSSION

In this chapter, we will dive into some discussion from our results. We review differences between the most fit and most accurate solutions, and how mutation rate increases the difference in their accuracy. We also investigate any potential biases in the PGA by using the different mapping functions.

Differences Between Most Fit and Most Accurate Solutions

In the results chapter, we find that all the representations have the tendency to have a larger difference in accuracy between the most fit solution and the most accurate solution as the mutation rate increases. We find the most fit solution tends to have a lower accuracy, implying the most fit solutions created with higher mutation rates make many small mistakes as compared to the most fit models with lower mutation rates that have high accuracy which frequency predict correctly.

Although our objective is to minimize loss, we also want a model with high accuracy. Since we find that higher mutation rates can decrease accuracy while maintaining similar loss, it is important to look at both the fitness and the accuracy when determining the optimal parameters for optimizing a binary prediction model using binary cross entropy loss, particularly the mutation rate.

PGA Bias

When investigating the results of the PGA, we found the weights for the PGA are lower than the BGA and VGA counterparts (Figure 8, page 38). We believe the lower weights may be

due to biases in the PGA's mapping functions. We investigate into all three map functions to try to determine if any bias could play into the PGA.

PGA1

PGA1 as demonstrated in Equation *4* on page 27 works by taking the count of characters and dividing by the length of the chromosome. Since the intermediate encoded value is found by dividing the count of a character by the length of the chromosome, there is a limited amount of value that can be that can be distributed among the encoded values. This bias is because the sum of all the counts of the characters cannot exceed the length. Since the intermediate encoded values are limited, that means our final encoded values are also limited, which may encourage smaller values as extremely high values take more characters to encode.

PGA2

PGA2 as demonstrated in Equation *5* on page 27 works by pairing up characters and dividing the count of the first character by the sum of the counts of the first and second characters as demonstrated in.

Unlike PGA1, the intermediate encoded value's precision is tied to the count of characters dedicated towards the encoded value. It takes more characters to encode precise values as more precise values takes a higher denominator; however, in many cases these intermediate values with higher denominators can be reduced into other intermediate values with smaller denominators (For example, 2/4 can be turned into 1/2), meaning that in this map function, intermediate encoded values that are reduced to more common fractions are more likely to occur.

We demonstrate the tendency to repeat more frequently occurring fractions for intermediate encodable values in Figure 24 which shows the possible weights generatable by PGA Map 2 and the frequency of their occurrence. We can see that there is a strong bias towards the most minimum and maximum values, followed by the median value. Given our range of -10 to 10, the PGA may find it easier to evolve weights like 3.33 or 5 as compared to other values of different precisions. This bias does not explain the bias towards lower weights; however, we note that it does encourage certain weights over others.

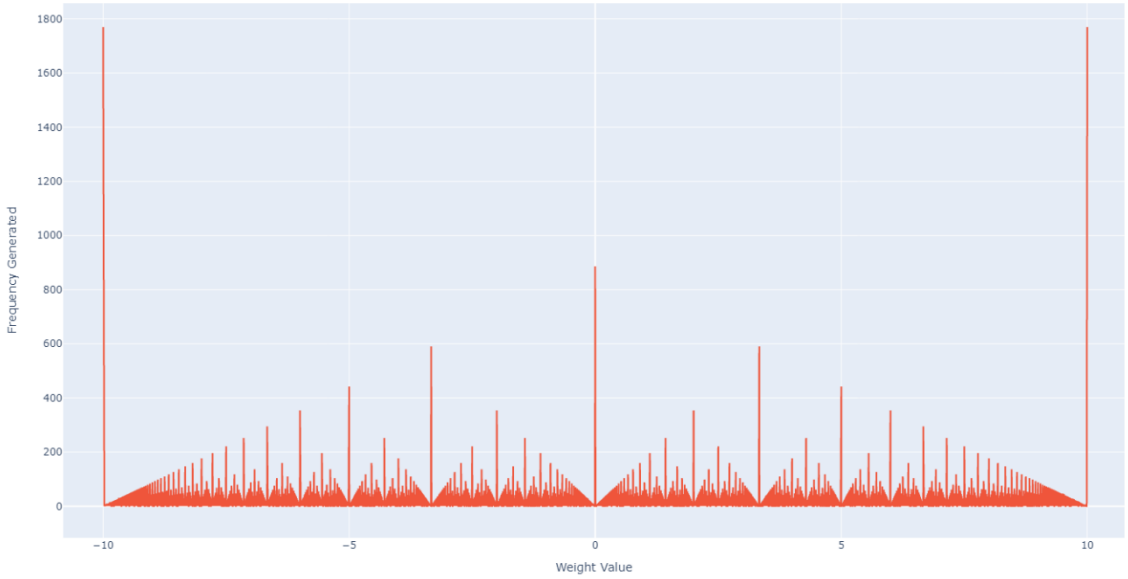

Figure 24: Bias of PGA2 towards weights represented by more commonly reduced to fractions

PGA3

Lastly, we go over any potential bias in PGA3. PGA3 as demonstrated in Equation *6* on page 27 operates by pairing up characters and taking the count of the less occurring character divided by the count of the more frequently occurring character. Like PGA2, the combination of different character counts may lead to equivalent intermediate encoded values, with a preference

towards intermediate encoded values that are commonly reduced to fractions. We demonstrate this in Figure 25 which demonstrates the possible values encodable on the x-axis and the frequency of those values on the y-axis. We can see that in addition to there being a bias towards intermediate encoded values that are reducible fractions, there is also a bias towards lower values.

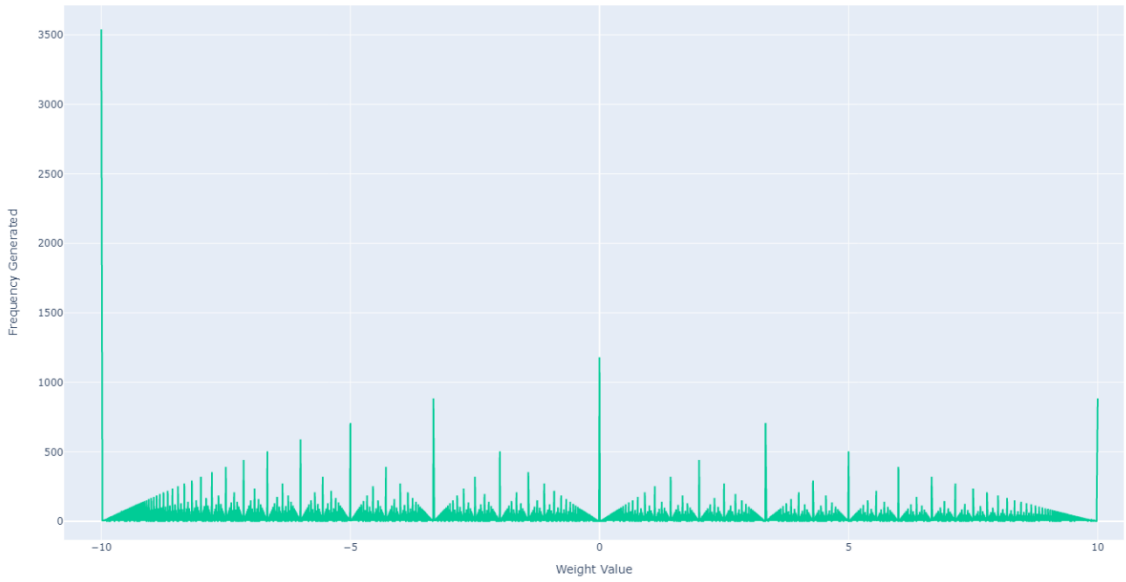

Figure 25: Bias of PGA3 towards weights represented by more commonly reduced to fractions and towards lower values

Since there is a bias below zero, that means that it is more likely when using PGA that the toggle genes will be below 0, meaning more weights will be disabled by default. Evolving weights with positive toggles are harder than positive toggles.

Averaging Effect of Crossover

Another potential reasoning behind why the PGA is biased towards lower weights may be due to crossover. The PGA has the tendency to average values during crossover (Wu & Garibay, 2002). The characters in PGA chromosomes tend to evenly distribute over the chromosome over time, so subsections of a chromosome have the tendency to be representative of the whole. When merging the subsections of the parents' chromosomes, we are left with what is effectively the weighted average of the frequencies of the selected parents.

CONCLUSION

In this thesis, we compared the ability to create binary prediction models using Genetic Algorithms (GA) with different problem representations on the Medicare Physical Therapist dataset. We compare the weights, fitness scores, and accuracy of the different representations and different parameters. We then discuss the difference between the most fit and most accurate individual, along with some potential biases in the PGA.

We utilized three different problem representations, the vector GA (VGA), binary GA (BGA), and the proportional GA (PGA). We find that all three representations can create models of similar accuracy; however, VGA tends to produce the most accurate on average. We do find that the most accurate individual was evolved by PGA2. Although the BGA has similar accuracy, it does not score as well as the VGA and the PGA, likely due to its more limited precision.

We notice that the most fit solution in runs with high mutation rate tend to have low accuracy but have similar loss scores as the most accurate solution from the same run, leading us to believe that higher mutation rate leads to evolving models that make many small mistakes, leading to lower loss scores but also lower accuracy. This observation leads us to stress the importance of evaluating both the loss and the accuracy when setting parameters for GAs optimizing binary prediction models using binary cross entropy.

The PGA also has notably lower weights than the VGA and BGA. We explore any potential bias in the PGA. We find all three mapping functions do have a bias; however, we only find possible biases towards lower values for PGA1 and PGA3, but not for PGA2. The original

work outlying the PGA by Wu and Garibay finds that the crossover in the PGA tends to average the selected parents, making extreme values harder to evolve (Wu & Garibay, 2002).

Despite not being the reasoning behind lower weights, we still investigate the bias from the mapping functions. PGA1 may have smaller values as weights are represented by what percentage of the chromosome their assigned letter uses, meaning there is a limited amount of weight distributable. PGA2 and PGA3 are likely to generate intermediate values more likely to be equivalent to other intermediate values before being mapped to the final encoding values, meaning certain intermediate encoded values like ½ and ¾ occur more frequently. The more frequent occurrence of certain intermediate encoded values leads weights associated with ½ and ¾ to be more likely to occur than weights associated with intermediate encoded values like $^{7}/_{8}$ or $^{9}/_{16}$. In addition to the bias for certain values, PGA3 also has the increased chance of encoding lower weights as the numerator is always the minimum value of the pairs of character counts.

We show that all three representations can produce high accuracy and low loss models with similar features; however, the weight values may be different in magnitude. We find high mutation rates can lead to evolution of most fit solutions with low fitness but also low accuracy. We also find biases in the PGA mapping functions that, although do not explain the lower weights, do demonstrate it's easier to encode certain values or combinations of values more than others.

REFERENCES

Altman, N. S. (1992). An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression. *The American Statistician, 46*(3), 175-185. doi:10.1080/00031305.1992.10475879

Bean, J. C. (1994). Genetic Algorithms and Random Keys for Sequencing and Optimization. *ORSA Journal on Computing, 6*(2), 154-160. doi:https://doi.org/10.1287/ijoc.6.2.154

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression Trees.* New York, New York, United States: CRC Press. doi:https://doi.org/10.1201/9781315139470

Burke, D. S., De Jong, K. A., Grefenstette, J. J., Ramsey, C. L., & Wu, A. S. (1998, December). Putting More Genetics into Genetic Algorithms. *Evolutionary Computation, 6*(4), 387-410. doi:10.1162/evco.1998.6.4.387

ChatterJee, S., Laudato, M., & Lynch, L. A. (1996). Genetic Algorithms and their Statistical Applications: an Introduction. *Computational Statistics & Data Analysis*, 633-651. doi:https://doi.org/10.1016/0167-9473(96)00011-4

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. doi:https://doi.org/10.1016/j.asoc.2007.02.009

Desbordes, P., Ruan, S., Modzelewski, R., Vauclin, S., Pierre, V., & Gardin, I. (2017). Feature Selection for Outcome Prediction in Oesophageal Cancer using Genetic Algorithm and

Random Forest Classifier. *Computerized Medical Imaging and Graphics, 60*, 42-49. doi:https://doi.org/10.1016/j.compmedimag.2016.12.002

Eshelman, L. J., Caruana, R. A., & Schaffer, J. D. (1989). Biases in the Crossover Landscape. *InProceedings of the Third International Conference on Genetic Algorithms* (pp. 10-19). Fairfax: Morgan Kaufmann Publishers Inc. doi:10.5555/93126.93134

Fernandez, A., Garcia, S., Luengo, J., Bernado-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. doi:10.1109/TEVC.2009.2039140

Fidelis, M. V., Lopes, H. S., & Freitas, A. A. (2000). Discovering Comprehensible Classification Rules with a Genetic Algorithm. *Proceedings of the 2000 Congress on Evolutionary Computation. 1*, pp. 805-810. IEEE. doi:10.1109/CEC.2000.870381

Fix, E., & Hodes, J. L. (1989). Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties. *International Statistical Review / Revue Internationale de Statistique, 57*(3), 238--247. doi:https://doi.org/10.2307/1403797

Gkisser, S. (1993). *Predictive Inference: an Introduction.* (S. Gkisser, Ed.) New York, New York, United States: Chapman and Hall/CRC. doi:https://doi.org/10.1201/9780203742310

Guvenir, A. H., & Erel, E. (1998, February 16). Multicriteria Inventory Classification Using a Genetic Algorithm. *European Journal of Operational Research, 105*(1), 29-37. doi:https://doi.org/10.1016/S0377-2217(97)00039-8

Jefferson, M. F., Pendleton, N., Lucas, S. B., & Horan, M. A. (1997). 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. doi:https://doi.org/10.1002/(SICI)1097-0142(19970401)79:7<1338::AID-CNCR10>3.0.CO;2-0

Kovacic, M., & Dolenc, F. (2016). Prediction of the Natural Gas Consumption in Chemical Processing Facilities with Genetic Programming. *Genetic Programming and Evolvable Machines, 17*, 231--249. Retrieved from https://link.springer.com/article/10.1007/s10710-016-9264-x

Min, S.-H., Lee, J., & Han, I. (2006, October). Hybrid Genetic Algorithms and Support Vector Machines for Bankruptcy Prediction. *Expert Systems with Applications, 31*(3), 652-660.

Ng, S. T., Skitmore, M., & Wong, K. F. (2008). Using Genetic Algorithms and Linear Regression Analysis for Private Housing Demand Forecast. *Building and Environment, 43*(6), 1171-1184. doi:https://doi.org/10.1016/j.buildenv.2007.02.017

Norat, R. (2020). Improving Usability of Genetic Algorithms through Self Adaptation on Static and Dynamic Environments. *University of Central Florida*. Retrieved from https://purls.library.ucf.edu/go/DP0023153

Orvosh, D., & Davis, L. (1994). Using a Genetic Algorithm to Optimize Problems with Feasibility Constraints. *Proceedings of the First IEEE Conference on Evolutionary*

Computation. IEEE World Congress on Computational Intelligence. 2, pp. 548-553. IEEE. doi:10.1109/ICEC.1994.350001

Stojanovic, B., Milivojevic, M., Ivanovic, M., Milivojevic, N., & Divac, D. (2013). Adaptive System for Dam Behavior Modeling Based on Linear Regression and Genetic Algorithms. *Advances in Engineering Software, 65*, 182-190. doi:https://doi.org/10.1016/j.advengsoft.2013.06.019

Tolles, J., & Meurer, W. J. (2016, August 2). Logistic Regression, Relating Patient Characteristics to Outcomes. *JAMA, 316*(5), 533-534. doi:10.1001/jama.2016.7653

Wolpert, D. H., & Macready, W. G. (1997, April). No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation, 1*(1), 67-82. doi:10.1109/4235.585893

Wu, A. S., & Garibay, I. (2002, June). The Proportional Genetic Algorithm: Gene Expression in a Genetic Algorithm. *Genetic Programming and Evolvable Machines, 3*, 157-192. doi:https://doi.org/10.1023/A:1015531909333

Wu, A. S., & Garibay, I. I. (2004). Intelligent Automated Control of Life Support Systems Using Proportional Representations. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 34*(3), 1423-1434. doi:10.1109/TSMCB.2004.824522

Wu, A., Liu, X., & Norat, R. (2019). A Genetic Algorithm Approach to Predictive Modeling of Medicare Payments to Physical Therapists. *The Thirty-Second International Flairs Conference.* AAAI Publications. Retrieved from https://www.aaai.org/ocs/index.php/FLAIRS/FLAIRS19/paper/viewPaper/18188

Wu, X., Kumar, V., Quinlan, J. R., Ghosh, J., Yang, Q., Motoda, H., . . . Steinberg, D. (2007, December 4). Top 10 Algorithms in Data Mining. *Knowledge and Information Systems, 14*, 1-37. doi:https://doi.org/10.1007/s10115-007-0114-2

Yu, H., & Wu, A. S. (2005). An Incremental Approach to the Proportional GA. *Proceedings of the 7th annual conference on Genetic and evolutionary computation.* Washington DC: Association for Computing Machinery. doi:10.1145/1068009