AUTOMATED MACHINE LEARNING: A BIOLOGICALLY INSPIRED APPROACH.

A Dissertation
Presented to
The Academic Faculty

By

Jason Zutty

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

December 2018

AUTOMATED MACHINE LEARNING: A BIOLOGICALLY INSPIRED APPROACH.

Approved by:

Dr. Aaron Lanterman, Advisor, Chair
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Gregory Rohling, Co-Advisor
Georgia Tech Research Institute
Georgia Institute of Technology

Dr. Jennifer Michaels
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Justin Romberg
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Mark Davenport
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. May Wang
School of Biomedical Engineering
Georgia Institute of Technology

Date Approved: October 29, 2018

If I have seen further, it is by standing on the shoulders of giants.

Isaac Newton

To my first advisor, Dr. Thomas Michaels, who helped me begin this journey; and to my grandmother, Dr. Roselyn Kessler, who inspired me to start it.

ACKNOWLEDGMENTS

First and foremost, I would like to thank Dr. Greg Rohling, without whom I believe none of this research would exist. You have been an excellent mentor to me, and I appreciate all the time you have spent working with me and challenging me. I think together we stumbled on a fascinating thread of research, and I look forward to continuing this work with you.

I want to thank Dr. Aaron Lanterman for stepping in to serve as my advisor to help me finish this long journey. I appreciate your careful reading and thoughtful revisions to my writing.

To Dr. Jennifer Michaels, Dr. Justin Romberg, Dr. Mark Davenport, and Dr. May Wang, I thank you for serving on my committee and for taking the time to understand my research.

I would also like to thank Dr. Gisele Bennett. Your guidance throughout my research has been instrumental. I appreciate the belief you had to help get EMADE started, both with our internal projects and our first sponsored application of EMADE with Chemical Companion.

I extend a special thank you to Marcel Sarzen. I am happy that we found such an appropriate application for EMADE in a space that is so impactful. I look forward to future collaborations and seeing where our work takes us. I would also like to thank Leanne West, for her help in matching our work together.

Next, I must thank all of the students in Dr. Rohling's and my Vertically Integrated Projects (VIP) class on automated algorithm design. You all have impressed me so much with how quickly you have learned such complicated material. We push you all very hard, and I am happy with how well you all have done. Of our students, I would like to extend special thank-yous to some that have had a significant impact on EMADE and its applications to real-world problems: Austin Dunn, Scott Heston, Milind Lingineni, Joseph Mosby, James Rick, and Roland Samuelson.

Completing this dissertation has been a long road, and I would not have made it without

the constant support of my friends and my family. Thank you to everyone who has stood by me through my nine years of graduate school.

Finally, I would like to thank the Georgia Tech Research Institute for the investment it has made in this research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Machine learning is a robust process by which a computer can discover characteristics of underlying data that enable it to create a model for making future predictions or classifications from new data. Designing machine learning pipelines, unfortunately, is often as much an art as it is a science, requiring pairing of feature construction, feature selection, and learning methods, all with their own sets of parameters. No general machine learning pipeline solution exists; each dataset has unique characteristics that make a particular set of methods and parameters better suited to solving the problem than others.

To respond to the challenge of machine learning pipeline design, the field of automated machine learning (autoML) has recently emerged. AutoML seeks to automate the often arduous work of a data scientist, so they can focus on the underlying meanings of the data and spend less time on the tedium of pipeline design and tuning.

This dissertation adapts and applies genetic programming to the newly emergent field of automated machine learning. Genetic programming enables the artificial evolution of an algorithm through a nearly infinite search space that otherwise requires a randomized search.

This dissertation shows that through the process of genetic programming, it is possible to produce machine learning pipelines, and the evolved pipelines can outperform those created by human researchers.

The original contributions of this dissertation are:

1. A unique data representation for efficient genome construction that increases the likelihood of successful evaluations.

2. A system of tiered evaluations that greatly accelerates the speed of the evolutionary process by reducing the number of genomes that must train and evaluate against a full dataset.

3. A methodology for supporting high-level building blocks, such as machine learning methods and signal processing functions, that enables genetic programming to compete with and evolve from human-created algorithms.

4. The Evolutionary Multi-objective Algorithm Design Engine (EMADE), an open-source framework for the automated creation of algorithms that automatically solves feature-domain, time-domain, and image-domain problems with better performance than algorithms created by the hands of experts.

5. Demonstration of EMADE's capabilities on different types of open machine learning problems, including:

   (a) The adult dataset problem, a feature classification challenge.

   (b) Dog behavior classification from accelerometer data, a time-domain classification problem.

   (c) Prediction of safe-working times for first responders, a feature regression problem.

   (d) Computation of water depth from LIDAR returns, a time-domain regression problem.

6. A fuzzy selection operator that uses a probabilistic interpretation of dominance in multi-objective space, often yielding higher-performing genomes in fewer evaluations than traditional selection operators.

7. An investigation into the strategic pairing of parents to produce better offspring than traditional evolutionary algorithms, which use random chance to produce their pairings.

CHAPTER 1
INTRODUCTION

This dissertation explores automated machine learning. This chapter briefly describes the concepts of genetic algorithms, multiple objective genetic algorithms, genetic programming, and vector-based genetic programming. After presenting these approaches, we enumerate the research contributions of this dissertation and outline the rest of the document.

1.1 Genetic Algorithms

A genetic algorithm works by simulating an evolutionary process to search for a set of values that can be input into a black box evaluator and yield an optimal solution (e.g. the largest or the lowest output that box can generate). A particular set of values that are input to the black box is called a *genome*, while the set of all possible genomes is referred to as the *search space*. The traditional representation for a genome is a list of values. Each one of these values is analogous to a particular *gene*. The output of the black box is taken to be an *objective score* for the corresponding input genome.

To simulate an evolutionary process, the genetic algorithm requires a set of genomes, which is called a *population*. The genetic algorithm assigns each genome in the population a *fitness* score that ties the genome's objective score, produced by the evaluation function, to the rest of the population with a measure of relative performance.

Figure 1.1 shows a diagram of the typical evolutionary loop. Each generation of the simulated evolutionary process, the genetic algorithm uses a *selection* process to choose the next generation of genomes based on their fitness scores. The genomes selected in this fashion are known as the *parents* of the next generation. The parents then exchange genes through crossover and have genes permuted through mutation.

An important step for most modern genetic algorithms is the computation of an *elite*

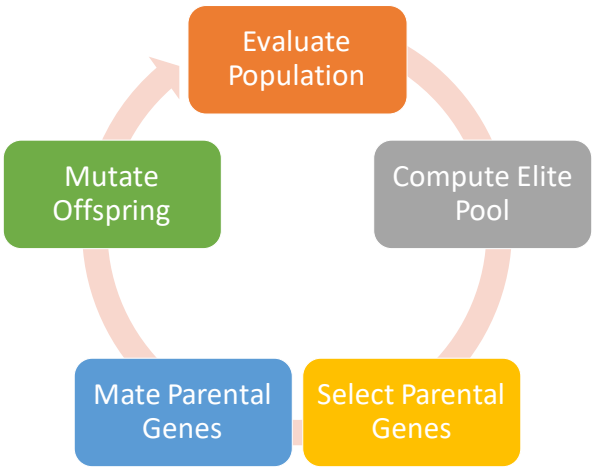


Figure 1.1: This diagram shows the high-level steps of the evolutionary process that repeat until the algorithm has converged.

pool. The elite pool acts like an archive to preserve solutions that have demonstrated leading objective scores. The presence of an elite pool helps the evolutionary process avoid cyclical patterns that can emerge when random processes cause successful individuals to fall out of the population.

Genetic algorithms excel when the *fitness landscape*, or mapping of genomes to objective space, is rough, meaning it is nonlinear and discontinuous. This rough objective space results in many local minima and maxima. In these rough environments, traditional gradient descent based optimization methods are likely to fail, getting stuck on a local optima unable to find other regions of the space that may contain better solutions.

A classic example problem for genetic algorithms is solving the one-max problem: Evolve a boolean vector of size d that has the maximum sum, i.e. is all ones. In this example we are manipulating a genome where each gene is either a 1 or 0. Let $d = 10$ and an example genome a be

$$a = [0, 1, 1, 0, 0, 0, 1, 1, 0, 0].$$

This genome has an objective score of 4. Let genome b be

$$b = [1, 0, 1, 0, 1, 0, 1, 0, 0, 0].$$

Genome b also has an objective score of 4. A mating operation of a single-point crossover between a and b works as follows. A point is chosen at random:

$$a = \Big[0, 1, 1, 0, 0, \Big| \; 0, 1, 1, 0, 0\Big],$$

$$b = \Big[1, 0, 1, 0, 1, \Big| \; 0, 1, 0, 0, 0\Big].$$

The two parents exchange genes at the crossover point:

$$c = [0, 1, 1, 0, 0, 0, 1, 0, 0, 0]$$

$$d = [1, 0, 1, 0, 1, 0, 1, 1, 0, 0]$$

Then the objective score for c is 3, and the objective score for d is 5. Note that through crossover alone, there is a limit on what the genetic algorithm may discover because traditional crossover locks all genes in their particular locations in the genome. This restriction means that if no individual in the population has a gene in a particular location, then one must be produced through mutation in order to obtain change. In the previous example, in the fourth position both a and b have a zero. No amount of random crossover between the two parents can produce a child with a one in the fourth position. Likewise, a and b also have zeros present in the sixth, ninth, and tenth positions. In this example, we compute the maximum objective score without mutation is the sum of the number of positions in all genomes that contain a one. Therefore, if we set up a crossover-only evolution with a population of $\{$a, b$\}$, then the maximum objective score of the optimization is six.

1.2 Multi-Objective Genetic Algorithms

While classical genetic algorithms are a robust technique for solving problems that can be expressed with a single objective function, most performance in real-world problems is not captured by a single objective. While tying a single objective score to a fitness value

is relatively straight forward, there are many ways to approach the problem in multiple objectives.

Most multiple objective genetic algorithms rely on a concept of *Pareto-optimality* for their selection methods. A genome is considered to be Pareto-optimal if it is *non-dominated* in objective space, meaning no other genome outperforms on all objectives. This concept can be expressed as follows: let a population be represented as $\boldsymbol{x} = [x_0, x_1, ..., x_{N-1}]$, where x_i represents a single genome in the population of N genomes. Let $f(x_i) = [f_0(x_i), f_1(x_i), ..., f_{M-1}(x_i)]$, where f_k is the k^{th} objective function of M objective functions. Then an individual x_i is Pareto optimal in a maximization scheme if

$$\nexists j \in 0...N-1 \mid f_k(x_j) > f_k(x_i), \forall k \in (0...M-1), \tag{1.1}$$

or in a minimization scheme if

$$\nexists j \in 0...N-1 \mid f_k(x_j) < f_k(x_i), \forall k \in (0...M-1).$$

Figure 1.2 illustrates the concept of Pareto-optimality in a two-objective minimization problem. The points in the plot represent performance of genomes in this objective space. Red points represent non-dominated solutions, and the red line that connects them represents the non-dominated frontier. Blue points represent dominated solutions, i.e. those that have any another point that outperforms them in both objectives. The bounded area between the frontier and the axes is tracked by the *hypervolume*.

In single-objective genetic algorithms we can track the performance of the optimization using the fitness of the best performing genome. With multiple objectives, we must instead use the hypervolume enclosed by the non-dominated surface, which shrinks with a minimization problem or grows with a maximization problem.

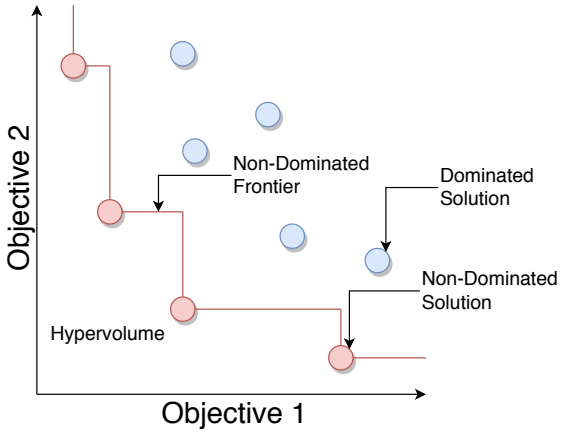


Figure 1.2: A 2-D illustration of Pareto-optimality when both objectives are being minimized.

1.3 Genetic Programming

Genetic programming (GP) differentiates itself from the broader class of genetic algorithms by its representation. While genetic algorithms manipulate the inputs to an evaluation method, genetic programming modifies methods themselves as well as the connections between them. Traditional genetic programming uses a tree structure as its genome. We refer to the nodes of these trees as *primitives*, which can be any function that consumes inputs and produces a single output. Historically, genetic programming uses low-level primitives such as arithmetic, trigonometric, and logical functions. We call the leaf nodes of the trees *terminals*, and they consist of inputs and scalar values that the primitives consume.

GP traditionally implements the tree structure as a lisp-like *parse tree*, which is a pre-ordered list representation. Each operator (or node) is followed by the inputs to that operator. For example, the equation:

$$x^2 + 2x + 1$$

would be expressed as:

$$[+, 1, +, *, 2, x, *, x, x].$$

Figure 1.3 shows the tree structure that implements this parse tree.

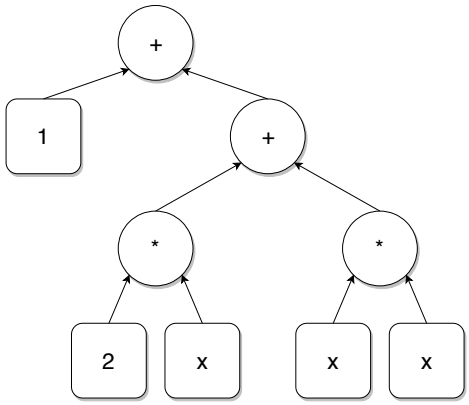


Figure 1.3: Tree structure that implements the equation $x^2 + 2x + 1$.

In genetic programming, crossover is the action of choosing random nodes in parents and swapping subtrees from those nodes between the parents. Mutation of genomes occurs by directly modifying the tree structure such as by removing primitives, changing primitives, changing terminals, or inserting subtrees.

Crossover between the earlier individual means selecting a random subtree (underlined):

$$x^2 + 2x + 1 = [+, 1, +, \underline{*, 2, x}, *, x, x]$$

and exchanging it with a random subtree from another individual (underlined):

$$x^3 - x + 5 = [+, 5, +, *, -1, x, \underline{*, x, *, x, x}],$$

which yields the following two children:

$$[+, 1, +, \underline{*, x, *, x, x}, *, x, x] = x^2 + x^3 + 1$$

$$[+, 5, +, *, -1, x, \underline{*, 2, x}] = 2x - x + 5.$$

Figure 1.4 shows this crossover example in tree structure format. The top two tree stuctures show the parents while the bottom two trees show their offspring. The subtrees exchanged are outlined in blue and green. Note that the tree in the top left is the same as the one in

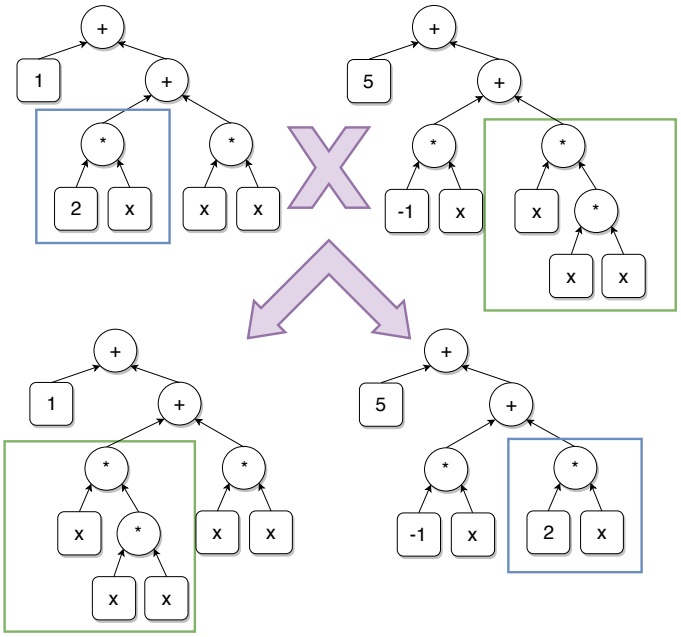


Figure 1.4: Crossover between $x^2 + 2x + 1$ and $x^3 - x + 5$.

Figure 1.3, but here we call attention to the subtree to be crossed over.

1.4 Machine Learning

Machine learning (ML) is a robust process by which a computer can discover characteristics of underlying data that enable it to create a model for making future predictions or classifications from new data. Machine learning can be either *supervised*, *semi-supervised* or *unsupervised*, depending on the number of observations of data that have an associated notion of *truth*. Truth is fed to the machine learning model for fitting and scoring purposes. In this sense, truth is the value the model should be predicting or classifying for a particular instance of data. Supervised machine learning is the most common, and occurs when every observation of data has an associated truth value. Unsupervised learning occurs when data is unlabeled, and usually consists of clustering methods to segregate instances into similar groups based on statistical properties of the data. Semi-supervised learning is when some instances of data contain truth and some do not.

Traditionally, supervised learning consists of two types of problems: *classification* and

regression. Classification is the family of problems where the model must distinguish data into discrete classes. Regression is the process for finding a relationship between features, allowing for the prediction of a continuous value.

Machine learning is broken up into a number of steps including data *preprocessing*, *feature engineering*, *model selection*, and *parameter optimization*. We refer to a machine learning *pipeline* as the point when the processed data enters the algorithm up to the output of a final prediction or classification.

Preprocessing is the action of preparing data for the learning process and involves imputation (i.e. replacing missing data with substituted values) or removal of data that was malformed while being captured. The preprocessing step is also where we divide the data into partitions. In this dissertation, we use three partitions referred to as *training*, *testing*, and *validation* data. Training data is used to fit machine learning models, while testing data is used to score the fitted models. Because genetic algorithms and genetic programming search methods are driven by the scores of the testing data, we withhold a third set of data to score the final models on data that they have not been exposed to previously. The preprocessing step may also consider the balance of observations in each of the three datasets and ensure that all three are statistically similar.

Feature engineering is the process by which raw data is transformed to find characteristics that allow for models to best discriminate between categorical data or identify trends in continuous data. This process can involve any combination of the following: the construction of features from data, the synthesis of new features from existing features, or the selection of a subset of features. For classification problems, the best feature space is one that best separates the classes. For regression problems, the best feature space is one in which some combination of features yields the highest correlation with truth data.

There are many different types of machine learning models that each have a set of strengths and weaknesses. Model selection is the choice of a model, or combination of models through *stacking* or *ensemble* techniques, that best adapts to the particular feature

space and truth data at hand. Stacking occurs when the prediction of one model is used as a feature for another. Ensemble techniques allow a number of models to each make a prediction, and disambiguate results from the set of predictions. An example ensemble technique can be something as simple as a majority vote.

Finally, parameter optimization is the last step of the design of a machine learning pipeline. Here, the parameters used for each step of the pipeline can be tuned to obtain the best performance for a particular pipeline against a targeted set of data. Techniques that automatically set and tune parameter pipelines are often referred to as *hyper-heuristics*.

1.5 Vector Based Genetic Programming for Automated Machine Learning

Traditional genetic programming (GP) only supports the use of arithmetic and logical operators on scalar features. In GP, these operators, referred to as primitives, are combined in tree structures representing the DNA of a program. When approaching an optimization problem from this frame of reference, there are several apparent issues. First, any solutions created by GP in this way must evolve from the most basic of building blocks. If we compare this to the evolution of humankind from single-cell organisms, this is a task that took nature almost four billion years to accomplish. It took approximately three billion years for multi-cell organisms to evolve from single-cell organisms, which represents 75% of evolutionary time on earth. Second, these low-level scalar features make it difficult to leverage the human-generated solutions that have already been discovered in many applications. For example, if one wanted to evolve improvements to something as simple as a k-nearest neighbor algorithm, it would be extremely difficult to construct a representation of it with only a standard set of operators: $(+, -, *, /, \&, |, !, <, >, ==)$. Finally, solutions produced by GP are usually dense, deep, tree structures. This makes them difficult to read and understand, preventing a human researcher from effectively learning from the results of the GP. Because of these limitations, it is uncommon to find solutions generated by GP that can outperform even the most basic human-derived machine learning functions.

When we use higher-level input features such as vectors and matrices, more advanced functions such as signal processing and machine learning functions can be combined with the standard set of operators. These functions, along with a set of objectives, enables GP to design readable, human-competitive algorithms directly from data. Allowing GP to produce the algorithms automatically is valuable in our current age of big data. Algorithm design can be challenging for a researcher who must choose from a host of feature engineering and machine learning techniques, and choose a set of hyper-parameters for each step. When looking at a problem, a skilled person will typically approach a solution with a limited set of techniques. Furthermore, it is difficult for a scientist to design an algorithm optimized for multiple objectives. GP can create non-domain-biased solutions and do so while optimizing for multiple objectives simultaneously. This new design paradigm can free up a researcher's time to steer the optimization towards specific solutions by analyzing the data and resulting programs. The research may also be inspired by the algorithms that the GP creates and pursue new courses as a result.

Many challenges must be addressed to successfully work with high-level primitives as we have described. To support machine learning functions as primitives, training and testing data used by each learner must be passed through the program. The same operations that were performed on training data must be performed on testing data at any given step. Strongly typed GP, which places restrictions on the tree structure based on the inputs and outputs of the primitives, enforces the rigidity required by the functional blocks we use. However, the standard data types available are too limiting; they do not allow data to flow naturally from primitive to primitive. We require a custom type to contain our data and wrapper methods that allow our primitives to understand this type. This allows the output of one block, whether it is a signal processor or a machine learner, to be an input to the next method. The container type must also be understood by evaluation functions, since this formulation requires the object to be both the input to and output from the entire program.

Despite the use of more advanced operator functions, we still leverage the fundamen-

tal attributes of GP (mating, mutating, and selection), and this means the evaluation of multitudes of individuals is still necessary. These individuals, however, are more computationally expensive to evaluate than those in traditional GP due to the cascading of machine learning functions and larger amounts of data. Generating solutions in a reasonable amount of time requires access to a computer cluster that can support memory-intensive processes.

This dissertation is a natural evolution of previous work. Past attempts at combining GP and machine learning have invoked a rigid search space, focusing on either feature construction or the optimization of an existing learner or set of learners. There has been little work performed that employs both of these approaches. When combined, there is significant added benefit to be gained from cascading the results of these methods.

We created the Evolutionary Multi-objective Algorithm Design Engine (EMADE) framework to implement automated algorithm design from high-level operators. This dissertation describes increases of EMADE's capability by seeking methods to reduce compute time and produce solutions in fewer evaluations.

1.6 Original Contributions

This dissertation shows that through the process of genetic programming it is possible to produce machine learning pipelines, and the evolved pipelines can outperform those created by human researchers.

The original contributions of this dissertation are:

1. A unique data representation for efficient genome construction that increases the likelihood of successful evaluations. Section 3.2 explains the details of this data structure.

2. A system of tiered evaluations that greatly accelerates the speed of the evolutionary process by reducing the number of genomes that must train and evaluate against a full dataset. Section 3.10 details how this system works.

3. A methodology for supporting high-level building blocks, such as machine learning methods and signal processing functions, that enables genetic programming to compete with and evolve from human-created algorithms. Sections 3.3 and 3.4 show how these high-level building blocks are assembled.

4. The Evolutionary Multi-objective Algorithm Design Engine (EMADE), an open-source framework for the automated creation of algorithms that automatically solves feature-domain, time-domain, and image-domain problems with better performance than algorithms created by the hands of experts. Chapter 3 shows the novel components that differentiate EMADE from other autoML frameworks.

5. Chapter 4 demonstrates EMADE's capabilities on different types of open machine learning problems, including:

   (a) The adult dataset problem, a feature classification challenge.

   (b) Dog behavior classification from accelerometer data, a time-domain classification problem.

   (c) Prediction of safe-working times for first responders, a feature regression problem.

   (d) Computation of water depth from LIDAR returns, a time-domain regression problem.

6. A fuzzy selection operator that uses a probabilistic interpretation of dominance in multi-objective space, often yielding higher-performing genomes in fewer evaluations that traditional selection operators. Section 5.1 walks through the methods and performance analysis of this operator.

7. An investigation into the strategic pairing of parents to produce better offspring than traditional evolutionary algorithms, which rely on random chance. Appendix A shows the results of this investigation.

Table 1.1: Capability by problem type

	Input Data Type		Problem Type	
Problem Name	**Feature**	**Time Series**	**Classification**	**Regression**
Adult Dataset	X		X	
Heat Stress	X			X
AGLogica		X	X	
LIDAR		X		X

1.7 Overview

Chapter 2 gives context by further explaining the concepts of genetic algorithms and genetic programming, as well as how they fit into the broader field of automated machine learning based on prior efforts. Chapter 3 introduces the EMADE framework and describes its unique capabilities. Chapter 4 shows the capabilities of the EMADE framework in a number of different problem classes as shown in Table 1.1. Chapter 5 presents the research contributions made in order to support EMADE, namely the capability to achieve the same or better results as traditional methods while performing fewer and/or cheaper evaluations. Chapter 6 summarizes our work and suggests further research to continue exploring the field of automated machine learning using genetic programming.

CHAPTER 2
BACKGROUND

2.1 Origins

The field of evolutionary computing grew out of John Holland's book *Adaptation in Natural and Artificial Systems*, published in 1975 [1]. Holland showed that the evolutionary process discovered by Darwin [2] can be translated into many applications using what is now known as the genetic algorithm [3]. Genetic algorithms have five key components: a method of representation, an evaluation function, an initialization procedure, genetic operators, and parameters [4, 5].

In 1992, John Koza gave life to the specific field of genetic programming (GP) with his book *Genetic Programming: On the Programming of Computers by Means of Natural Selection* [6]. In this book, Koza showed that GP was able to produce successful results to problems in a variety of fields. GP is a specific case of genetic algorithms, where each individual in the population is a representation of a computer program [3].

The key differentiation between GP and genetic algorithms is that a genetic program is represented as a parse tree, rather than a vector of numbers. In his introductory book [6], Koza defined genetic operations on the tree structure, representing the trees as LISP expressions. Aside from the representation of an individual, almost all of the techniques from genetic algorithms can be applied to the evolutionary process of GP.

2.2 Fitness

It is natural for evolutionary algorithms to live in a multi-objective space, such as the simultaneous maximization of true positives and the minimization of false negatives. Fitness functions, however, produce a single scalar value to represent the aptitude of the individual

relative to all other individuals in the gene pool. Many fitness functions are surveyed in Fonseca and Fleming's *An Overview of Evolutionary Algorithms in Multi-Objective Optimization* [7]. A key concept used in multi-objective fitness functions is the evaluation of relative performance based on Pareto dominance, which stems from the notion that not all of an individual's objectives may be simultaneously improved. By rewarding individuals that are Pareto-dominant, we may drive the optimization toward the true Pareto-optimal frontier. Equation 1.1 defines Pareto dominance.

While Pareto dominance is the main driver of most fitness functions, there are many different strategies for handling the case of individuals that are equal in Pareto rank. Most of these strategies rely on population-based clustering in objective space. These strategies tend to have different methods, but follow the common thread of using a *crowding distance* measure, rewarding individuals in less crowded regions of objective space. The logic behind rewarding the more distant solutions is that those individuals contribute more phenotypic diversity to the population, which adds important genotypic diversity.

The non-dominated sorting genetic algorithm (NSGA), introduced in *Multi-Objective Optimization using Non-Dominated Sorting in Genetic Algorithms* [8], was one of the first genetic algorithms to emphasize the Pareto optimal multi-objective space. NSGA is useful for pushing the Pareto front by assigning its highest fitness to the first non-dominated front, and decreasing fitness with increasing dominance levels. In their paper introducing NSGA-II [9], Deb et al. point out that NSGA has several weakness, including a high computational load, a lack of elitism, and a need for a sharing parameter. NSGA-II introduced a fast non-dominated sort algorithm and diversity preservation through crowding distance sorting.

The Pareto Envelope-based Selection Algorithm (PESA) [10] and PESA-II [11] compute crowding factors based on the amount of Pareto-optimal solutions that occur in the same hyper-box in objective space. Those with fewer are selected over those with more when using a binary tournament. PESA-II builds on this strategy by introducing the concept of regions to PESA.

Zitzler et al. [12] introduced an improved version of the Strength Pareto Evolutionary Algorithm (SPEA) [13]. This fitness scoring method assigns each evaluated chromosome a score that corresponds to the number of other chromosomes in the population that dominate it. An individual on the Pareto front would be assigned a score of zero. The authors also found that SPEA2 had the highest performance in high-dimensional objective spaces compared to the performances of SPEA, NSGA-II, and PESA. The methods that had the best overall performance were SPEA2 and NSGA-II.

Some more recent algorithms for fitness include novelty search [14] and MOEA/D [15]. Fitness can also be computed based on an individual's contribution to the hyper-volume of the Pareto surface. Algorithms that use this concept include SMS-EMOA [16] and HypE [17].

Rohling [18] proposed a hypercube distance (HCD) scaling algorithm that scales fitness values based on where a solution exists in objective space. The algorithm works with two thresholds for each objective function: an "achievable" level and a "goal." Each individual's precomputed fitness is modified according to the formula

$$i_{\text{scaled distance}} = i_{\text{distance}} \times \frac{1+\alpha}{\alpha + \left(\sum_{k=0}^{K} HCD[k]\right)^{\frac{1}{p}}}, \tag{2.1}$$

where i_{distance} is crowding distance of individual i (larger is better), α and p are constant weight parameters (suggested to be 0.5 and 2, respectively, which manifest as euclidean distance), K is the number of objectives, and $HCD[k]$ is

$$\begin{cases} 1, & f_k \geq \text{achievable}_k \\ 0, & f_k \leq \text{goal}_k \\ \left(\dfrac{f_k - \text{goal}_k}{\text{achievable}_k - \text{goal}_k}\right)^p, & \text{otherwise} \end{cases} \tag{2.2}$$

in a minimization scheme. In Equation 2.2, f_k represents the objective score, achievable$_k$ is the threshold for an achievable result, and goal$_k$ is a desired objective value. Section 3.7 illustrates HCD in more detail.

2.3 Selection

After assigning fitnesses to each individual in a population, the next step of the evolutionary process is to select the parents for the next generation. A selection algorithm translates each fitness score to a probability of selection for mating. There are three main forms of selection used in GP: fitness proportionate selection (also known as roulette wheel selection), tournament selection, and truncation selection.

Roulette wheel selection assigns each chromosome a probability of selection based on their fitness relative to the rest of the population. If $f(x_i)$ represents the fitness score of one individual in a population of N individuals, then the probability of selection for that individual is

$$p(x = x_i) = \frac{|f(x_i)|}{\sum_{k=0}^{N-1} |f(x_k)|}.$$

In tournament selection, selection occurs in rounds, where in each round k individuals are randomly chosen. The individual with the best fitness score will proceed to the next round. In a binary tournament, $k = 2$. A truncation selection sorts the population in order of fitness scores and then chooses the first M of the N chromosomes, where the M individuals are those with the best fitness. This truncation example assumes descending fitness where a higher fitness is better than a lower one. In a problem setup where a lower fitness is better than a higher one, truncation would either take the last M chromosomes or sort in ascending order.

Roulette wheel and binary tournament selection are more "bio-inspired" than truncation selection, as the latter does not assign any probability of mating to those below the fitness threshold. In tournament selection, every individual except that with the absolute

lowest fitness score has some probability of mating. With roulette wheel selection, every individual has some probability of mating. Also, unlike binary tournament and roulette wheel selections, there is no randomness in a truncation algorithm. Because randomness is important for a genetic algorithm to search properly, we usually pair truncation with another selection method.

2.4 Algorithm Examples

Most papers on fitness schemes also pair them with particular selection methods. In an effort to make these fitness and selection pairings more clear, we will walk through two algorithms in more detail: NSGA-II and SPEA2. This dissertation leverages both of these algorithms to manage different parts of the evolutionary process.

2.4.1 Non-dominated Sorting Genetic Algorithm II

NSGA-II [9] uses a binary tournament in which dominance is given by

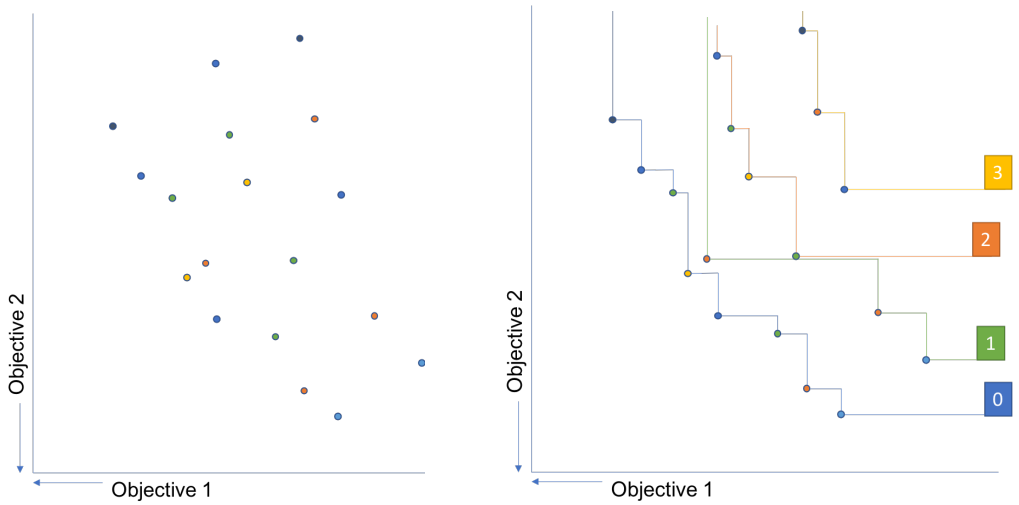
$$\begin{aligned} &i \prec_n j \quad \text{if} \quad (i_{\text{rank}} < j_{\text{rank}}) \quad \text{or} \\ &((i_{\text{rank}} = j_{\text{rank}}) \quad \text{and} \quad (i_{\text{distance}} > j_{\text{distance}})). \end{aligned} \tag{2.3}$$

Here, $i \prec_n j$ means that individual i dominates j; this occurs if i_{rank}, the Pareto (non-dominated) rank, is lower than j_{rank}, which means that i closer to the non-dominated front. In the case of a tie in rank, $i \prec_n j$ if i_{distance}, the crowding distance of i, is greater than j_{distance}, indicating individual i may fall in a less crowded region.

This selection process works as follows. First, the algorithm combines the previous generation's parents and their offspring into a population. Next, the algorithm sorts the population into non-dominated ranks. This sort operates as follows:

1. Initialize rank $r = 0$ and copy all individuals in the population to a list p.

2. Initialize a new list of lists nd, indexed by rank r.



(a) A sample objective space.

(b) The objective space partitioned into non-dominated ranks.

Figure 2.1: Demonstration of applying NSGA-II to an objective space.

3. Identify the non-dominated individuals in the population list p.

4. Move the non-dominated individuals to list $nd[r]$ so that they are no longer in p.

5. If there are still individuals in p, increment r by one and go to 3.

Figure 2.1 illustrates this non-dominated sort. Figure 2.1a shows an example objective space. In this example, we wish to minimize both objectives. The ideal solution the problem this objective space represents would be the origin, i.e. (0, 0). Figure 2.1b shows this same objective space after computing non-dominated ranks. The blue line labled 0 shows the non-dominated frontier. In the list representation of the population, the sort organizes the individuals in ascending order of Pareto rank, i.e. [Rank 0, Rank 1, Rank 2, Rank 3] for this example.

Within each rank, NSGA-II sorts by crowding distance. Listing 2.1 shows the pseudo-code computation of crowding distance for a given individual. Within the NSGA-II, while a lower rank is better than a higher rank, a higher crowding distance is better than a lower crowding distance. The crowding distance serves to estimate the average side length of the hypercube that is formed around each individual in objective space when using its closest

neighbors as vertices.

Code Listing 2.1: Pseudo-code for crowding distance computation in NSGA-II

```
# Crowding distance is computed by rank, so we iterate through each rank
                                        individually.
for rank in ranks:
  L = len(rank)
  for individual in rank:
    # Initialize the crowding distance for each individual.
    individual.distance = 0
  for objective in objectives:
    # Sort current rank by objective score in ascending order.
    rank = sort(rank, objective)
    # Force individuals with min and max of objective to have highest
                                       distances so that they will beat out
                                        other individuals in the same rank.
                                        These individuals have no neighbor
                                       on one side.
    rank[0].distance = inf
    rank[-1].distance = inf
    # Iterate through remaining L-2 individuals to increment their
                                       distances.
    for i in range(1, L-1):
      # Add the normalized distance between the current individual's
                                       neighbors on this objective
      rank[i].distance = rank[i].distance + (rank[i+1].objective - rank[
                                       i-1].objective)/(rank[-1].objective
                                       - rank[0].objective)
```

NSGA-II truncates the sorted population to the desired population size. For clarity, if the desired population size is N, that yields N parents each generation. From these N parents we produce N children. NSGA-II then operates on a population of $2N$ to sort and

truncate back down to the next generation of N parents.

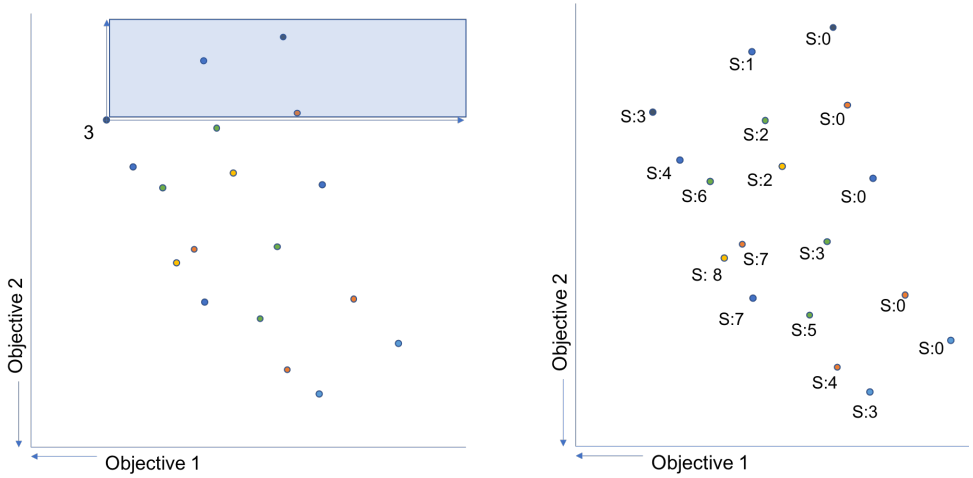
Now, the N parents are put through a binary tournament where we select two parents N times (with replacement). For each selection, we compare the two randomly selected parents first by rank (favoring the lower) and then by crowding distance (favoring the higher). The resultant N winners are then crossed over and mutated to become the N offspring. Section 2.5 describes the mating and Section 2.6 describes mutation.

2.4.2 Strength Pareto Evolutionary Algorithm 2

Rather than utilize a Pareto rank determined by a non-dominated sort, the rank used by SPEA2 for a given individual comes from the number of individuals in the population that dominate the individual. The computation of this rank begins by computing a strength for each individual. The strength is the number of individuals in the population that the given individual dominates. Figure 2.2a shows a visual representation of this computation. In the figure, the individual of interest dominates the blue shaded area of objective space in this minimization problem. The individual receives a score of three, because three individuals exist in this dominated area. Figure 2.2b shows the resultant strengths of each individual in the population.

Next, SPEA2 uses the computed strengths for each individual to compute a rank for each individual. For each individual, the sum of the strengths of the individuals in the region of space that dominate the given individual produces the rank score. Figure 2.3a shows this computation of rank for a given individual. Figure 2.3b shows the ranks for all individuals in the population. Because no individuals dominate the Pareto-optimal solutions, they receive a rank of zero.

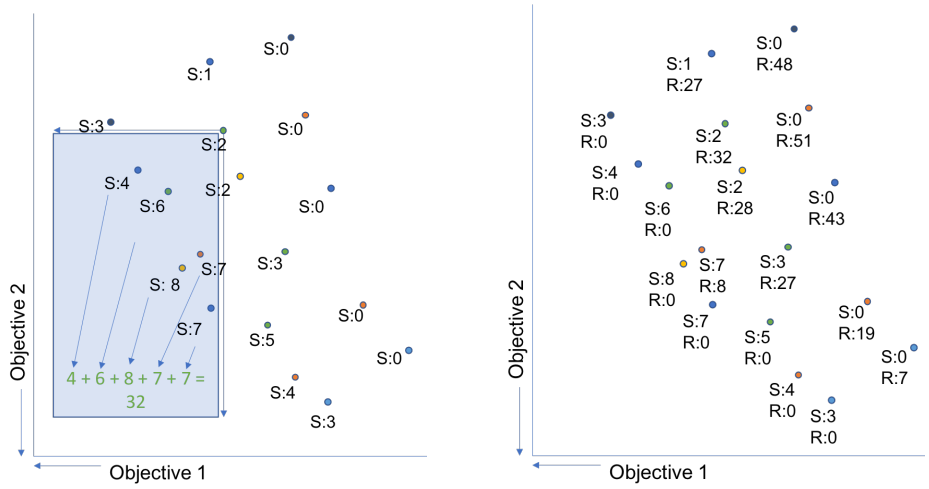
Next, to prevent ties on rank, SPEA2 adds a density estimation to each rank. Equation 2.4 shows the computation of this density metric. This equation relies on a selection of the parameter k. σ_i^k is the euclidean distance to the k^{th} nearest neighbor of individual i. A common choice of k is $k = \sqrt{2N}$, where $2N$ is the size of the competing population,



(a) Computation of strength for a single individual.

(b) Strength computed for each individual in the population.

Figure 2.2: Demonstration of computing strength for SPEA2.



(a) Computation of rank for a single individual.

(b) Rank computed for each individual in the population.

Figure 2.3: Demonstration of computing rank for SPEA2.

i.e. N parents and N offspring. A smaller density metric indicates an individual is in a less crowded area of objective space. This density metric is also always less than one. This means the addition of density will never push an individual to a higher rank. Equation 2.5 combines rank and density to create the fitness, for each individual, that drives selection:

$$D_i = \frac{1}{\sigma_i^k + 2}, \tag{2.4}$$

$$F_i = R_i + D_i. \tag{2.5}$$

Like NSGA-II, we truncate the population of size $2N$ to N individuals for the next generation. The truncation simply keeps the N individuals with lower ranks by sorting in ascending order. We then perform N binary tournaments with replacement to select the N parents for the next generation.

2.5 Mating

The standard operator for crossover in GP is the single-point crossover, proposed by Koza in *Genetic Programming: vol. 1* [6] as a modification of a standard technique used in genetic algorithms. In this method, a node is selected from each tree and all information from the corresponding subtrees are exchanged between the two chromosomes. Figure 2.4 shows an example of single-point crossover.

In Beadle and Johnson's *Semantically Driven Crossover in Genetic Programming* [20], a new method of crossover is proposed that ensures the two subtrees are not semantically equivalent. This prevents creating two new individuals that accomplish the same tasks as their parents. A similar theory is used by Ishibuchi and Shibata [21], where instead of comparing genotypic diversity directly (defined as the structure and elements of the trees), methods are proposed in selection based on maintaining phenotypic diversity (defined as relating to the objective scores of each individual).

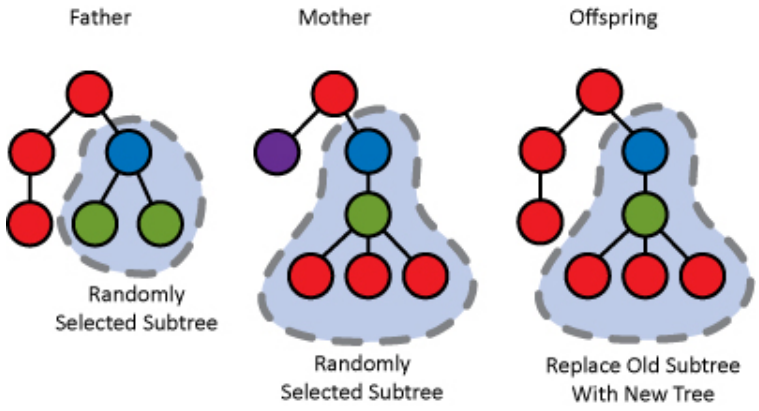


Figure 2.4: Creation of a child from two parents using single-point crossover. [19]

2.6 Mutation

There are several different mutation operators available to a GP optimization, including insertion, shrinking, and replacement. All three operations were proposed by Koza [6] and taken from genetic algorithms. While the operators traditionally operated on vectors, Koza rephrased them to operate on trees. "Insertion" is the creation of a new randomized subtree and inserting it at some point into the chromosome. "Shrinking", is the action of removing a subtree, and replacing it with one of its leaves. "Replacement" is changing a single node in the chromosome.

2.7 Hybridization: GP and Machine Learning Methods

While GP has not been used to directly operate on both machine learning and signal processing algorithms simultaneously, it has been used in conjunction with existing machine learning and signal processing methods individually, often achieving significant performance gains over the base algorithms. Using GP to perform feature construction or selection for machine learning is a well-researched field. There have been numerous approaches to allow the evolution of optimal feature sets.

H. Guo et al. [22] used GP with basic operators to generate features for machine learning classifiers. The authors compared the results of an artificial neural network (ANN) and

a support vector machine (SVM). For both types of classifiers, the features generated by GP showed significant improvements over the standard signal processing techniques alone.

Holladay et al. [23] introduce vector-based GP, a concept we use heavily in this dissertation. Sections 3.2, 3.3, and 3.4 detail how we leverage vector-based GP in this research. In this formulation of GP, primitives act on vectors of inputs, rather than scalars. We combine this with a notion from Vera et al. [24], which used GP on vector-based time series data with the last value of the series representing the prediction. This provides a starting place from which to target combining both signal processing methods and machine learning functions with traditional primitives.

Streater [25] used GP to evolve feature construction programs from basic primitives for inputs to machine learners, and evolved parameters for the SVMs he used for classification. The results were scored for fitness based on the performance of several machine learning algorithms on the constructed feature set. Rather than working with scalar coefficients or derived features, the inputs to the genetic program were six matrices, each of which represented a different channel of an image. However, high-level primitives were not used in his instruction set; instead, the instruction set consisted of simple mathematical operators: plus, minus, multiply, divide, sine, cosine, square root, and log. Finally, rather than include the learners in the evolutionary tree, they were used solely for evaluation.

L. Guo et al. [26] applied GP to the problem of feature extraction in electroencephalographic (EEG) signals. Their system consumes a feature vector created using a discrete wavelet transform (DWT), and evolves new features computed from the input set. The final set of features is sent to a k-Nearest Neighbors (kNN) classifier. However, the GP was only performed on scalar elements of the feature vector with the operators of plus, minus, times, divide, log, and square root. While L. Guo et al. were able to show significant improvement over the kNN classification by itself, treating the DWT coefficients as scalars limited the traction they achieved.

Al-Sahaf et al. [27] developed a hybrid algorithm that uses a nearest-neighbor classifier

to map GP output to a class label. This model was applied to a variety of health-related datasets from the UCI Machine Learning Repository [28]. While this approach showed improvements over naïve Bayes, SVM, and kNN applied directly to the data, the search space of evolved algorithms is limited by the small set of operators: addition, subtraction, multiplication, and division.

Lee and Tong [29] combined GP with the autoregressive integrated moving average (ARIMA) filter to improve time series predictions. The hybrid methodology used the form $\hat{y}_t = \hat{L}_t + \hat{N}_t$, where the predicted signal $\hat{y}_t$ is the sum of the predictions of the ARIMA filter $\hat{L}_t$, which forecasts the linear component of the data, and the GP $\hat{N}_t$, which forecast the nonlinear residuals. For the nonlinear portion, they found a lower dependence on the size of available datasets, and were able to achieve higher quality results than other machine learning techniques, including SVM and ANN. While they did combine traditional machine learning techniques with GP, they did so in a limited fashion, applying it with a standard set of basic operators and only focusing on errors made by a linear model.

Ravisankar et al. [30] proposed a hybrid of GP and machine learning to predict the failures of dotcom companies. The authors divided the problem into its two key pieces, feature selection and classification. A hybrid algorithm used a different machine learning technique for feature selection than for classification. However, the authors did not consider feature construction. Cascaded combinations of both selection and construction allow for more complicated solutions.

Zhang et al. [31] combined GP with SVMs to produce credit scoring models, using a divide and conquer approach like Lee and Tong. GP was used to develop a set of classification rules for the data. Any residual data not assigned a class by GP would be classified using an SVM. The authors were able to produce hybrid algorithms that had higher accuracies than any standalone machine learners. However, they enforce a limitation on the type of machine learner used and restrict GP to only produce classification rules. Both of these human derived choices can themselves be open to optimization.

In Sherrah et al.'s work [32], the genetic program was allowed to not only create and select features but also choose from three different classifiers as well. When compared to standard machine learners, error was reduced in eight out of the nine tested datasets. However, once again, the genetic program was limited by scalar features, basic primitives, and a rigid structure.

2.8 Non-GA/GP Heuristic Methods

Hassan and Cohanim [33] showed genetic algorithm performance against a newer search space heuristic: particle swarm optimization. While they show the quality of results is similar, the space they use is limited. PSO moves candidate solutions in space to find optima, but with GP, solutions do not have locations that can be acted on, just the resultant Pareto space. While PSO is a powerful state-of-the-art technique, it is not a well suited alternative to solve the problem of algorithm creation.

Simulated annealing is a popular alternative to GP, and one strongly suggested by Steven Skiena in his *Algorithm Design Manual* [34]. Simulated annealing [35] uses random exploration of a search space with a decaying probability of accepting worse solutions over time. The idea of changing probabilities of accepting inferior solutions over time can be applied to adaptive mating and mutation rates throughout the evolutionary process.

2.9 AutoML

In recent years, the field of automated machine learning has begun to take shape. One of the the leaders in this field is the tree-based pipeline optimization tool (TPOT) [36]. TPOT was developed at the University of Pennsylvania Epistasis labs to automatically combine scikit-learn [37] machine learning functions using DEAP[38]. As Section 3.1 shows, the work presented in this dissertation uses these same tools. Our work began prior to the development of TPOT. While utilizing similar packages, our solution allows for both a more flexible genotype and phenotype. Sections 4.1 and 4.5 draw comparisons between

our research and TPOT.

Other AutoML frameworks using similar concepts include Auto-Weka [39], auto-sklearn [40], and RoBO [41]. These frameworks focus on tuning hyper-parameters of machine learning algorithms through techniques such as Bayesian optimization, as opposed to the bio-inspired approach in this thesis. The more flexible bio-inspired approach enables a richer search space of possible algorithms. While these frameworks are all promising, they all work directly on feature data, rather than raw data sources, which greatly limits their applicability to challenging problems in the time, image, or video domains.

In industry, the field of AutoML has been growing as well with companies like H2O.ai [42], RapidMiner [43], and DataRobot [44] introducing automated methods of developing and tuning models. One of the most recent and powerful entrants in this space is Google, with their Cloud AutoML platform [45]. Unlike other competitors, this framework is able to work with raw data types and train and tune more complex models. It is specifically designed for the field of computer vision, and based off the concept of transfer learning, which modifies pre-trained models from similar applications for the task at hand.

CHAPTER 3
INTRODUCTION TO EMADE

3.1 Creating The EMADE Framework

Although Sections 2.7 and 2.9 show that combining traditional machine learning techniques with GP has been accomplished in various applications, the current state of the art has not yet opened up the entire algorithm creation process to GP. We enable the modification of features and cascade the results of machine learning operations by using GP to create new solutions, thus addressing the limitations of the works presented in Sections 2.7, 2.8, and 2.9, and enabling technical contributions in Chapters 4 and 5.

The Evolutionary Multi-objective Algorithm Design Engine (EMADE) framework is built on top of DEAP (Distributed Evolutionary Algorithms in Python) [38], an existing GP framework. We use DEAP for many reasons, including its active development, distributed evaluation, and plug-and-play design. The DEAP framework enables us to drop in new definitions of individuals, primitives, terminals, and genetic operators, allowing support for a new data type that ties together training and testing data. EMADE combines boolean logic, arithmetic operators, signal processing, machine learning algorithms, and other primitives to create high-level algorithms that are capable of outperforming human-derived solutions.

To facilitate quick application to new problems, an XML input file is used to begin an EMADE optimization process. In the XML file, it is possible to define which data files, evaluation functions, mating, mutation, and selection operators are being used. The input file also allows the definition of any parameters associated with the operators, such as data types, or probabilities.

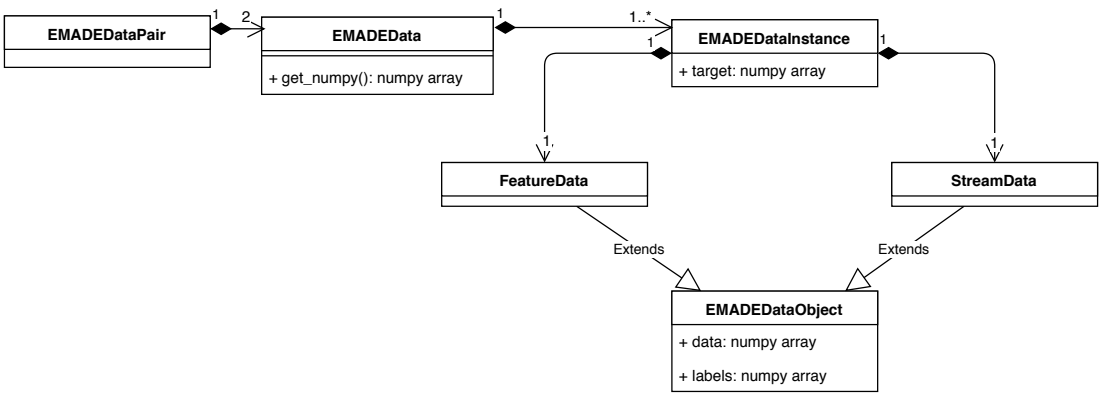


Figure 3.1: UML diagram of the data structure within EMADE.

3.2 Data Structure

EMADE uses strongly typed genetic programming, which enforces a set of rules during the tree creation, mating, and mutation processes. To build complex classification and prediction algorithms with EMADE, we added the ability to pass data between signal processing functions and machine learning functions in any order.

To handle these transactions, we created a Python class that stores various representations of the features and class information for each dataset. These data objects are paired together in a new class that supports training and testing data. This allows the same signal processing methods to be applied to the training and testing data before and after going through the machine learning operations. EMADE also allows the truth data, i.e. the correct values of the test data, to be directly compared with the results of the test data at the end of the program.

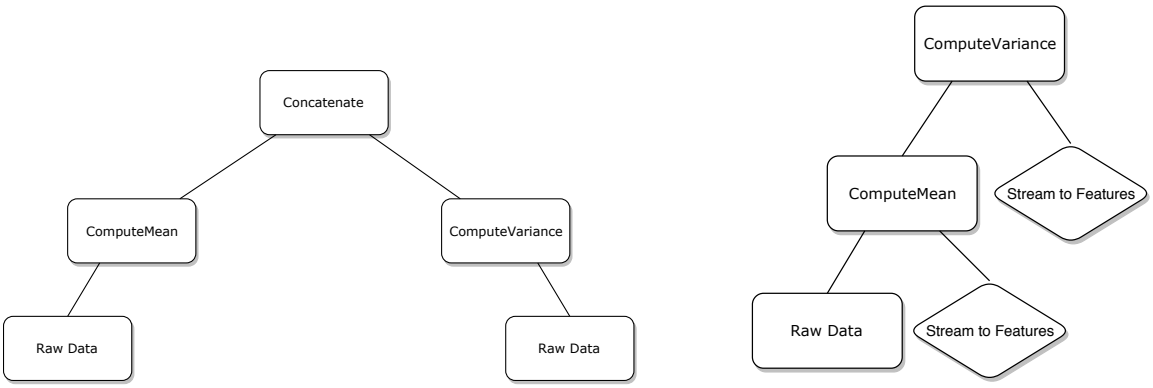
Figure 3.1 shows the UML (Unified Modeling Language) diagram for the data structure used in EMADE. The highest level representation of the data is an EMADEDataPair. This single object stores two EMADEData objects, one representing the training data and one representing the testing data. Each EMADEData object is a collection of EMADEDataInstance objects. Each EMADEDataInstance represents one observation in a dataset. The instance supports several different containers for storing data that makes up a single observation.

The first is a FeatureData container. This container holds the features for the observation. We define features as descriptive values, where the order of the variables holds no meaning. For example, the age or weight of a person could be considered features. Similarly, we could consider the average and standard deviation of a signal to be features as well. The second type of container is StreamData. Stream data is data where the order has specific meaning. Examples include one-dimensional time series waveforms, two-dimensional arrays representing images. or three-dimensional arrays representing movies. While FeatureData objects are 1-Dimensional, StreamData can be N-dimensional. The final component of the EMADEDataInstance is the target. The target is the supervised value of the instance, meaning the item we are designing our classifier or regressor to predict based on the features.

Implementation of a dual representation of data in EMADE allows for a more *evolvable* tree structure than treating stream and feature data separately. We say a tree structure is more evolvable if the probability of a destructive mating or mutation is lower. A destructive operation is one where the child becomes an invalid program or algorithm, usually resulting in an error when executed.

To understand how this dual representation makes our trees more evolvable, consider solving a signal processing problem without it. If we begin by taking in a waveform for each instance and feed that directly into a machine learning algorithm, we will have poor performance due to the time-varying nature of the data, i.e., for all but the most trivial signals, the machine learners will not be able to make sense of the data without constructed features. To make a useful algorithm, we need derived features. If instead the data is loaded into a stream container for the raw data in our dual representation, the features object will remain empty, and the algorithm will be known to be fatal and hence not require evaluation. Being able to identify algorithms prior to evaluation that will not be successful dramatically speeds up the evolutionary process.

Now suppose that the algorithm has a properly constructed feature from an averaging



(a) Without dual representation. (b) With dual representation.

Figure 3.2: An example of feature construction without and with a dual representation of the data.

algorithm, returning the mean for each instance. If we want to combine our feature with another, e.g., a variance from each instance, we will need to take in the same raw signal elsewhere in the tree and then join the resultant features together with a concatenate operation. Figure 3.2a illustrates this algorithm as a tree structure. Although this example is relatively simple, as the evolution proceeds, the trees will become increasingly complex, resulting in algorithms that construct more derived features. The result of this complexity is a structure that relies heavily on concatenation nodes. Should a concatenation node be mutated to an algebraic node such as an addition, as Figure 3.3 shows, the tree is at risk for dimensionality errors.

The probability of a fatal mutation grows linearly with the number of constructed features, i.e., the number of concatenation nodes. Our dual representation solves this problem by pairing feature construction and concatenation together. Figure 3.2b shows how the construction in Figure 3.2a looks in our dual representation. Section 3.4 covers how our representation leverages an additional parameter that indicates to the primitive how to process its input data.

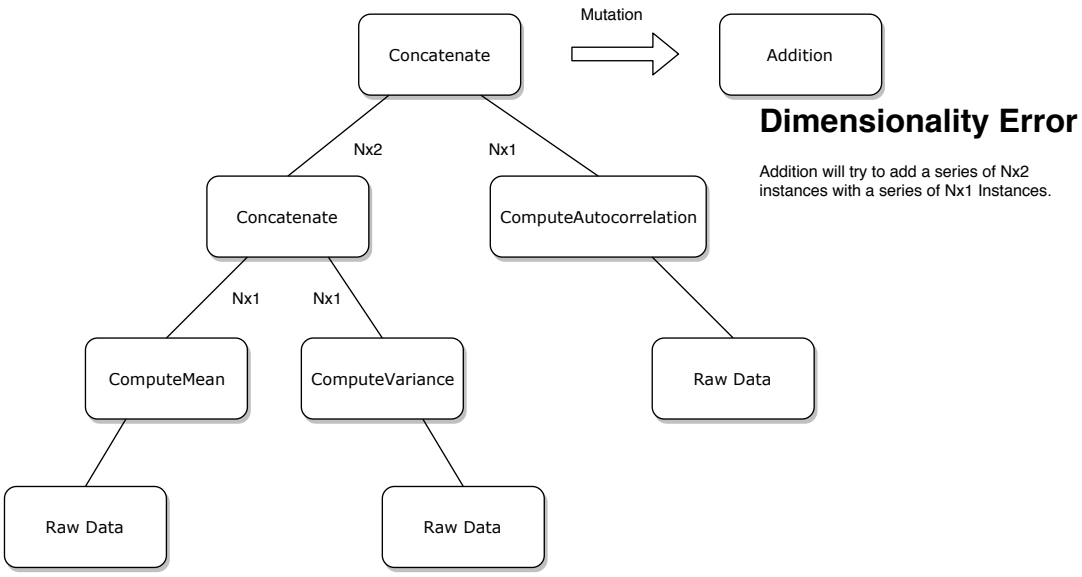


Figure 3.3: A mutation of the root concatenation node to an algebraic type of node.

3.3 Wrapping Machine Learning Functions

To use higher-level machine learning functions from existing Python toolboxes such as Scikit [37] and Keras [46], as well as several hand-coded functions, we developed a method of wrapping these primitives to incorporate them in a DEAP tree structure. Each primitive must act on pairs of training data and testing data. The wrapping function must be able to accept the input in a paired format, and produce the output in the same format. Traditionally, most of these functions from toolboxes act directly on Numpy arrays. Numpy is a computationally efficient array and mathematical library for Python.

The general structure for wrapping a machine learning function is as follows: let x_{train} be the training data, which is an $n \times m$ feature matrix, and let x_{test} be the testing data, an $l \times m$ feature matrix, where m is the number of features and n and l are the number of training and testing sample points, respectively. Let the machine learning function being wrapped be f_{ml}, which produces a classifier or predictor:

$$f(x_1, x_2, ...x_m) = f_{\text{ml}}(x_{\text{train}})(x_1, x_2, ...x_m).$$

Then, for the matrix we can produce a vector of estimates produced by the wrapped machine learner f_{ml}:

$$f(x_{\text{test}}) = f_{\text{ml}}(x_{\text{train}})(x_{\text{test}}) = \hat{y}_{\text{test}}.$$

We can also produce the same vector for the prediction on the trained data:

$$f(x_{\text{train}}) = f_{\text{ml}}(x_{\text{train}})(x_{\text{train}}) = \hat{y}_{\text{train}}.$$

Each vector is appended as an additional feature column to the respective matrices so their results can be built upon by further methods in GP. The returned data pair then comprises

$$x'_{\text{train}} = [x_{\text{train}}, \hat{y}_{\text{train}}]$$

and

$$x'_{\text{test}} = [x_{\text{test}}, \hat{y}_{\text{test}}],$$

with dimensions $n \times (m+1)$ for x'_{train} and $l \times (m+1)$ for x'_{test}. The number of features must remain consistent between training and testing data, so if the data pair is consumed by another machine learning primitive, the resulting function produced by f_{ml} will be applicable to x_{test}. Therefore, the associated class data for x'_{test} is also assigned to be $\hat{y}_{\text{test}}$. This ability for one machine learning function's predictions to be used as a derived feature in another machine learning model is known as *stacking* in the machine learning community.

Another benefit of the wrapping of the machine learning functions is the ability to store the produced classifier or predictor. Each resultant f_{ml} can be *pickled* out of the program. Pickling is a method of preserving objects in Python. The pickled data is labeled by the parameters of the wrapped machine learner, as well as a *hash* of the x_{train} data it operated on. This hash comes from a series of mathematical operations that perform a one-way compression of the data into a much smaller representation (in our case 256 bits). The mathematical operations have an extremely low probability of producing the same hash

for different data. Before rerunning the training of the machine learner, the file system is inspected for this stored object. If it exists, it is used in place of recomputing the machine learner, which saves a considerable amount of time, as subtree crossover mutation often produces new algorithms with branches that contain machine learners that have already been trained and evaluated exactly.

Code Listing 3.1 shows a template for wrapping machine learning functions. The code works as follows: first, the learner description is converted to a *base estimator*. A base estimator is an instantiated object used by the scikit API that can later be fit to data, or once fit, can predict values from data. Next, the target values that the model is trying to predict are extracted from the training data in the data pair. Features are then extracted from both the training data and testing data. The model is fit on the trained features and target values from the training data. The trained model is used to predict the target values of the testing features. Finally, the newly created targets are copied to a new feature column to be leveraged by further learners.

Code Listing 3.1: Template for Wrapping a Machine Learning Function

```
def single_learner(data_pair, learner):
    """
    Generic wrapper for a machine learning function
    That uses a scikit-learn estimator to fit on the
    features found in the training data of a data pair,
    and predicts on the features found in the testing data
    of a data pair.
    """
    # Get the underlying base estimator to use
    base_estimator = get_scikit_model(learner)

    # Deep copy of data so no modifications unintentionally
    # propagate to other branches of the tree structure
    data_pair = copy.deepcopy(data_pair)
```

```
# Grab the training features as a numpy array where each
# row represents an instance
training_data = data_pair.get_train_data().get_numpy()

# Extract the truth data associated with each instance
target_values = np.array(
    [inst.get_target()[0] for
     inst in data_pair.get_train_data().get_instances()])

# Check for multiple target values
target_value_check(target_values)

# Using the scikit-learn API, fit the estimator to the
# training features and the truth data associated with them
base_estimator.fit(training_data, target_values)

# Grab the features associated with the test data
testing_data = data_pair.get_test_data().get_numpy()

# Use the fitted estimator to predict values for the
# test features.
predicted_classes = base_estimator.predict(testing_data)

# Store the predicted values on the target of each instance
# in the test object of the data pair, this will be used
# for scoring
[inst.set_target([target]) for inst, target in
  zip(data_pair.get_test_data().get_instances(), predicted_classes)]

# Make the predictions a feature through use of the
# "make feature from class" method that adds it as a new feature
# column,  but then restore the truth data for the training data.
```

Table 3.1: Machine Learning Functions Implemented in EMADE as Primitives.

Classification	Regression	Ensemble
k-Nearest Neighbor Classifier	k-Nearest Neighbor Regressor	Adaboost
Support Vector Machine Classifier	Support Vector Machine Regressor	Bagged Learner
Decision Tree Classifier	Decision Tree Regressor	ExtraTrees
Random Forest Classifier	Random Forest Regressor	XGBoost [47]
Naive Bayes Classifier	Gradiant Boosting Regressor	LightGBM [48]
Logistic Regression Classifier		
Gaussian Mixture Model Classifier		
Best Linear Unbiased Predictor		
Orthogonal Matching Pursuit		
KMeans		
Stochastic Gradient Descent		
Passive Aggressive Classifier		

```
    # Set the self-predictions of the training data
    trained_classes = base_estimator.predict(training_data)
    [inst.set_target([target]) for inst, target in
      zip(data_pair.get_train_data().get_instances(), trained_classes)]

    data_pair = sm.makeFeatureFromClass(data_pair, name=learner.
                                        learnerName)
    # Restore the truth data to the training data for fitting
    # future learners.
    [inst.set_target([target]) for inst, target in
      zip(data_pair.get_train_data().get_instances(), target_values)]
    return data_pair
```

Table 3.1 lists the current set of machine learners that EMADE supports. To ensure that these machine learning methods can be optimized, each wrapped learner exposes the parameters it consumes to EMADE.

Machine learning primitives in EMADE operate only on data collected in the FeatureData containers of each instance, ignoring any data stored in StreamData parts of the instance.

Code Listing 3.1 shows this feature extraction through the get_numpy() function called on the training and testing data to return the features as a numpy array.

3.4 Wrapping Signal Processing Functions

Unlike traditional machine learning algorithms, in EMADE, signal processing primitives are not restricted to operate solely on feature data. Instead, each function takes in a parameter that specifies one of three modes: stream to stream, features to features, or stream to features. In the first two modes, each signal processing algorithm runs on each instance in both the testing and training datasets, altering the appropriate structure, be it stream or features. In the third mode, the signal processing algorithm creates a new set of features x' from the stream data $x(t)$. x' is appended to the existing x feature set.

Figure 3.4 presents various primitives operating on a given instance in EMADE. The first row represents the starting point for the instance of data, in this case a noisy waveform. The first method is a low pass filter (LPF) that operates in stream to stream mode. This filters the noisy waveform and replaces the stream data with the result. Next, we apply a fast Fourier transform in stream to features mode, which constructs coefficients from the stream data (that is now a denoised version of the original data), and places the resultant coefficients in the features bucket. The last method is a cumulative sum in features to features mode. Like the LPF, this primitive operates in place, computing a cumulative sum on the features bucket and replacing the features bucket with the result. Note that both buckets are carried from primitive to primitive through the tree structure.

Most signal processing functions in EMADE operate on each instance in the dataset independently. Code Listing 3.2 shows the general form of a signal processing function. The first step in wrapping a signal processing function is to correct input parameters to make the primitive as robust as possible. Robustness is important since the evolutionary process can produce a wide variety of parameters that can lead to errors in the underlying signal processing function, wasting time spent on evaluating the tree. If we can correct a

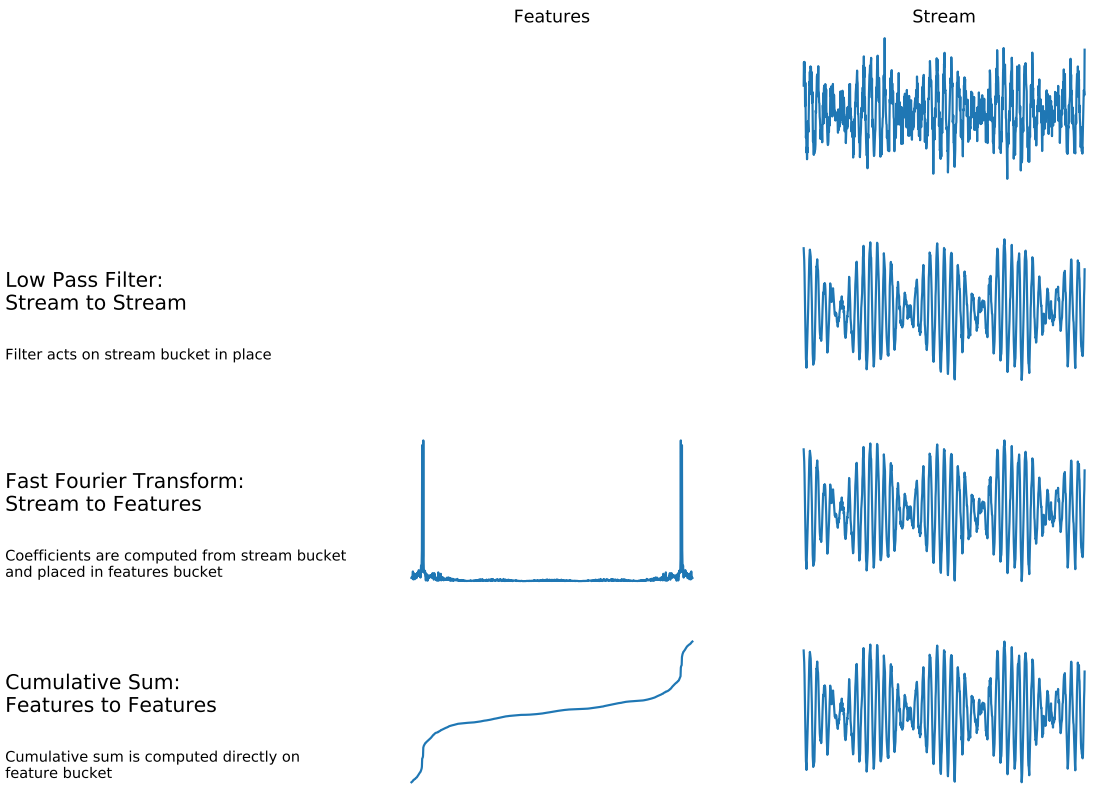


Figure 3.4: An example of how primitives interact with stream and features buckets for the algorithm: cumulativeSum(FFT(LPF(data, stream_to_stream), stream_to_features), features_to_features).

parameter to be within acceptable bounds, then we can potentially save valuable evaluation time. For example, if the function cannot process a negative number as a parameter, a simple correction would be to take the absolute value of the parameter.

After we correct parameters, we apply the function independently to the training data and the testing data. We first iterate on each dataset separately, then we iterate on each instance in that dataset. For each instance, we retrieve the data to be operated on from the appropriate container, as denoted by the mode of the primitive. We then run the function we are wrapping on the retrieved data, and assign it back to the container that the mode denotes. Table 3.2 shows the signal processing functions that EMADE currently supports.

Code Listing 3.2: Template for Wrapping a Signal Processing Function

```
def my_signal_processing_template(data_pair,
  param_1, param_2, mode=FEATURES_TO_FEATURES):
    """
    Psuedo-code for wrapping a signal processing
    function. This underlying example function has two parameters
    that are exposed to the wrapper.
    """
    # Step 1 correct parameters for robustness, e.g., if the wrapped
    # function expects positive inputs correct_parameter could be
    # the absolute value function
    param_1 = correct_parameter(param_1)
    param_2 = correct_parameter(param_2)

    # Initialize where the data will be temporarily stored
    data_list = []

    # Iterate through train data then test data
    for data_set in [data_pair.get_train_data(),
        data_pair.get_test_data()]:
        # Copy the dataset so as not to destroy original data
```

```
instances = cp.deepcopy(data_set.get_instances())
# Iterate over all points in the dataset
for instance in instances:
    # Based on the mode of the function,
    # decide which bucket to get data from
    if mode is FEATURES_TO_FEATURES:
        data = instance.get_features().get_data()
    elif mode is STREAM_TO_STREAM or STREAM_TO_FEATURES:
        data = instance.get_stream().get_data()

    # Now that we have the appropriate data,
    # apply the function we are wrapping
    data = run_function(data, param_1, param_2)

    # Based on the mode of the function,
    # decide where to put data back
    if mode is FEATURES_TO_FEATURES:
        instance.get_features().set_data(data)
    elif mode is STREAM_TO_STREAM:
        instance.get_stream().set_data(data)
    elif mode is STREAM_TO_FEATURES:
        old_features = instance.get_features().get_data()
        new_features = np.concatenate(
                        (old_features.flatten(),
                        data.flatten())
                        )
        # Set new labels for the generated features,
        # e.g., labels may be names of features, or time indices
        data_labels = generate_data_labels()
        new_labels = np.concatenate((
                        instance.get_features().get_labels(),
                        data_labels
                    ))
```

Table 3.2: Signal Processing Functions Supported in EMADE as Primitives.

Windowing Functions	Filters	Transforms	Math Functions
Hann	Averaging	Discrete cosine transform	Autocorrelation
Hamming	Difference	Fast Fourier transform	p-Norm
Tukey	Kalman	Discrete wavelet transform	Root mean square
Cosine	Wiener	Principal component analysis	Sum
Lanczos	Savitzky-Golay	Independent component analysis	Cumulative sum
Triangular		Sparse principal component analysis	Product
Bartlett		Linear predictive coding	Cumulative product
Gaussian		Empirical cumulative distribution function	Absolute value
Bartlett Hann			Log
Blackman			Arcsine
Kaiser			Arccosine
Planck Taper			Arctangent
Nuttall			Sine
Blackman Harris			Cosine
Blackman Nuttall			Tangent
Flat top			Exponential
			Cross correlation

```
                instance.get_features().set_data(
                            np.reshape(new_features, (1,-1)),
                            labels=new_labels
                            )
        new_data_set = EMADEData(instances)
        data_list.append(new_data_set)
    # Build EMADEDataPair
    data_pair = EMADEDataPair(train_data=data_list[0],
                              test_data=data_list[1])
    gc.collect(); return data_pair
```

Depending on the method or the mode of operation, the dimensionality of the data can change, both in terms of number of observations, and number of features. For this reason, the passing of training data and testing data throughout the tree in tandem is important.

3.5 Other Primitives

Table 3.3 shows a variety of other functions found in EMADE, including a number of image processing techniques implemented from OpenCV [49]. The table also includes

Table 3.3: Other Functions Supported in EMADE as Primitives.

Clustering	Affinity propagation	Mean shift	Dbscan	Spectral
	K-means	Agglomerative	Birch	
Feature Selection	K-Best	Percentile	Fpr	Fdr
	Generic Univariate	Fwe	Varience Threshold	
Image Processing	Minimum to zero	To uint8	To uint8 scaled	To float
	To Float normalize	Edge detection Canny	Corner detection Harris	Corner detection min eigenval
	Highpass Fourier ellipsoid	Lowpass Fourier shift	Highpass Fourier shift	Highpass Fourier Gaussian
	Highpass Fourier uniform	Highpass unsharp mask	Highpass Laplacian	Highpass Sobel derivative
	Lowpass filter median	Median blur	Lowpass filter average	Blur
	Lowpass filter Gaussian	Lowpass filter bilateral	Lowpass Fourier ellipsoid	Lowpass Fourier Gaussian
	Lowpass Fourier uniform	Threshold binary	Threshold to zero	Morph erosion rect
	Morph erosion ellipse	Morph erosion cross	Morph dilate rect	Morph dilate ellipse
	Morph dilate cross	Morph open rect	Morph open ellipse	Morph open cross
	Morph close rect	Morph close ellipse	Morph close cross	Morph gradient rect
	Morph gradient ellipse	Morph gradient cross	Morph tophat rect	Morph tophat ellipse
	Morph tophat cross	Morph blackhat rect	Morph blackhat ellipse	Morph blackhat cross
	Contours all	Contours min area	Contours max area	Contours convex concave
	Contours min length	Contours max length	Contour mask	Contour mask min area
	Contour mask max area	Contour mask convex	Contour mask min length	Contour mask max length
	Contour mask range length	Contour mask min enclosing circle	Contour mask max enclosing circle	Contour mask min extent enclosing circle
	Contour mask max extent enclosing circle	Contour mask range extent enclosing circle	Contour mask min aspect ratio	Contour mask max aspect ratio
	Contour mask range aspect ratio	Contour mask min extent	Contour mask max extent	Contour mask range extent
	Contour mask min solidity	Contour mask max solidity	Contour mask range solidity	Contour mask min equ diameter
	Contour mask max equ diameter	Contour mask range equ diameter	Threshold n largest	Threshold n largest binary

some feature selection and clustering methods from scikit and other sources. While we do not go into great detail about these methods, some of them are leveraged in Chapter 4 in applications of EMADE.

3.6 Mating and Mutation

EMADE supports multiple mating methods, including single-point crossover, ephemeral-only crossover, headless chicken crossover, and headless chicken ephemeral crossover. Single-point crossover is leveraged directly from DEAP, while ephemeral-only and headless chicken mating operators are custom-implemented genetic operators for EMADE. An ephemeral-only crossover is a mating method we developed for the specialization of exchanging regeneratable terminals between individuals, rather than entire subtrees as in traditional crossover. Figure 3.5 shows an example of ephemeral-only crossover. This method of mating is our way of supporting tuning, making this technique closer in spirit to genetic algorithms than to genetic programming, as it is only transferring parameters.Two individuals that undergo an ephemeral-only crossover will have their general structures preserved. Headless chicken crossover is a single-point crossover with one parent being a randomly generated tree. Headless chicken ephemeral crossover is our ephemeral-only crossover operating with one parent being a randomly generated tree.

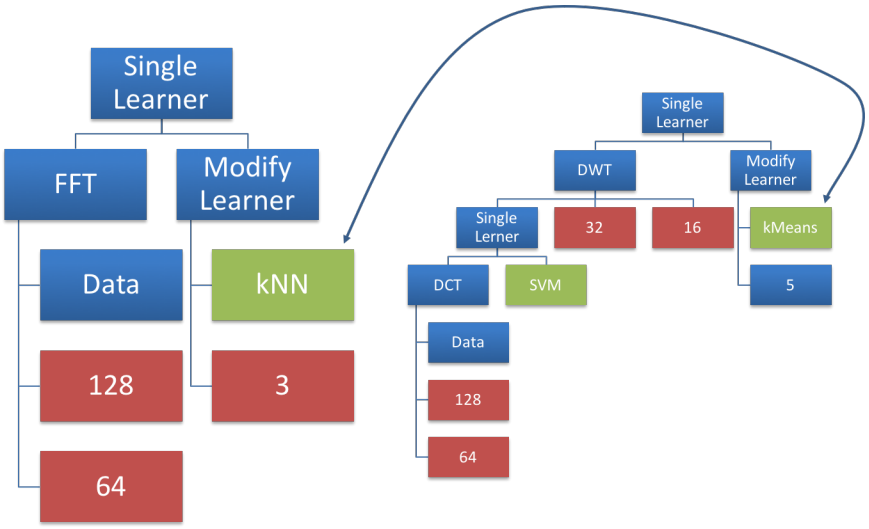


Figure 3.5: An example of an ephemeral only crossover. Note that only ephemerals of like types can be exchanged, such as a scalar (pictured in red) or a machine learning parameter (pictured in green).

Both headless chicken operators can be used as a litmus test for the effectiveness of genetic programming at solving a particular problem. For example, if headless chicken crossovers (randomness) are producing higher quality solutions than traditional single-point crossover, then we can assume that useful information is not being exchanged through crossover [50].

The mutation methods EMADE leverages are insertion, ephemeral, node replacement, uniform, and shrink. All five are implemented from DEAP. An insertion mutation randomly adds a new subtree to a location in the tree structure. An ephemeral mutation randomly regenerates an ephemeral constant, which is a terminal in the tree. A node replacement randomly changes one primitive to another. A uniform mutation is similar to an insertion, but rather than adding a subtree to the individual, it replaces a random subtree with a newly created one. A shrink mutation randomly removes a subtree, replacing the entire subtree with one of its inputs.

For the shrink mutation (also known as the node removal operator) to function correctly, we require an added objective representing parsimony, e.g., the depth of the tree, or the number of elements in the tree. Parsimony is essential because an algorithm that per-

forms as well as another should be considered to be dominant if it is less complex. While this objective does not directly affect performance, it does help support Occam's razor, "entities must not be multiplied beyond necessity," or more colloquially "other things being equal, simpler explanations are generally better than more complex" [51]. Together, a parsimony objective and shrink operator help to control bloat. Without the emphasis on parsimony, the genomes in EMADE would continue to grow unchecked throughout the evolutionary process. This bloat would increase the time it takes to evaluate each genome as the optimization progresses, as well as increase the probability of crossing over inactive subtrees (or latent DNA in our biology metaphor), which increases the probability of destructive operations. Chapter 4 shows some examples of bloat that appear in the application of EMADE.

3.7 Selection

Selection is performed using NSGA-II as described in Section 2.4.1, with the exception that crowding distances are modified prior to selection by applying the hypercube distance (HCD) scaling algorithm proposed by Rohling [18], described in Section 2.2. HCD scaling allows for the acceptance of an individual in a more crowded region if that region is more interesting.

This procedure serves to steer the selection process towards regions of interest in objective space. The scaling factor in our usage is bounded between $\frac{3}{1+2\sqrt{K}}$ (occurring when the individual lies outside of the achievable region) and 3 (which occurs when the individual lies inside of the goal region). Here, K is the number of objectives. Figure 3.6 shows a contour plot of how the scale factor changes based on an individual's location in objective space.

In EMADE, an elite pool is maintained and used to create a gene pool in each generation by appending any evaluated offspring, performing selection, and recomputing the elite pool. Unlike the offspring selection process (which uses an NSGA-II scheme), a strength

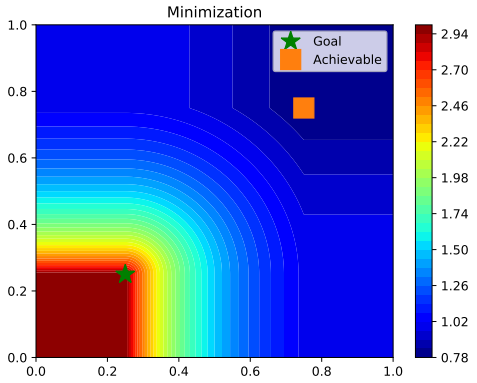


Figure 3.6: A plot of HCD in a two objective minimization example.

Pareto evolutionary algorithm (SPEA2) [12] truncation scheme is used to select the elite pool. Section 2.4.2 explained the details of the algorithm.

In the context of EMADE, the fitness of each individual is computed as $F(i) = R(i) + D(i)$. Here, $R(i)$ is the raw fitness score of individual i and $D(i) = \frac{1}{\sigma_i^k + 2}$, where σ_i^k is the distance to the kth nearest data point to i. The gene pool is sorted in ascending order by fitness, since SPEA2 is a minimization scheme. After the SPEA2 sorting, the pool is truncated to the N_{elite} lowest (best) elements. These elements form the elite pool.

The crowding distance in the elite pool calculation is not scaled using the HCD operator as it was for our modified NSGA-II. This can result in a wider coverage of the objective space in the elite pool than is desired by the thresholds set for each objective. The advantage, however, is more genetic diversity being preserved from generation to generation, even if it has a low probability of mating.

3.8 Hashing Evaluated Individuals and Subtrees

Facility Layout Optimization Using Simulation and Genetic Algorithms [52] showed that re-computation of individuals can be avoided efficiently through the use of a hash table. The EMADE framework uses a hash string representation (parse tree) of each created tree with an SHA-256 hash. EMADE stores the objective score vector of the individual with its

associated hash. When a new individual is created through mating, mutation, or generation, its string representation is hashed to determine if it already exists in the hash table. If it does, the values are pulled from the table, and it is not sent for evaluation. This trades off the relatively inexpensive cost of hashing each new genome for the time intensive cost of reevaluating an already computed individual.

EMADE also hashes results of machine learning training (in effect, subtrees). Before training a machine learning algorithm, the input feature data is hashed along with the learner type and parameters. If this hash has already been written, the trained machine learner is read from disk. This hashing method is significantly cheaper than retraining a learner on the same set of features. It also helps as subtrees are exchanged in mating, meaning trained and evaluated branches are reused across multiple individuals.

The expected benefit b_i that hashing learner i provides can be expressed as $b_i = r_i t_i$, where r_i represents the probability that the trained model will be used again on the same data, with the same model parameters, and t_i is the time it took to train model i. We also must consider limiting our cache to a particular size S. We can now phrase our caching system as a 0/1 knapsack problem where we wish to maintain a set j that maximizes $\sum_j b_j$ such that $\sum_j s_j < S$, where s_j is the disk size of learner j.

For time-series and feature-based problems, we do not cache results for signal processing functions. For the most part, the expected benefit of each method is significantly lower than that of machine learning functions, since signal processing functions do not have to be fit prior to application. While most machine learners have compact representations on disk, most signal processing methods in EMADE have roughly the same memory footprint as their input data, making caching a low-yield operation due not only to the limitations of storage, but also the amount of time required to write and read the cached information. Often, the write and read times are more expensive than the time to execute the functions in memory. In the future, as we move to solve more complex application problems in the imagery and video domains, we may find caching these functions might be more worthwhile.

3.9 Handling Large Datasets

DEAP handles its distributed computing through another Python package, SCOOP (**S**calable **Co**ncurrent **O**perations in **P**ython) [53]. SCOOP spins up a number of workers among a set of hosts. Because evaluating an individual can be memory intensive due to large amounts of data and complex primitives, having multiple workers on a single machine can quickly consume all of the available memory. To avoid this, each process's memory usage is monitored, and it is terminated if it exceeds a specified amount. SCOOP workers do not respawn; therefore, each SCOOP worker must start a separate process to run the evaluation. The original thread watches the newly created thread's memory usage, and terminates it if necessary. Any individual that consumes an intolerable amount of resources is given a poor fitness score to avoid reproduction in the future.

3.10 Tiered Datasets

Some of the challenges in working with machine learning functions include the memory and processing time required to preform classification or prediction. Both resources increase exponentially with the size of the datasets. Evaluating hundreds of thousands of individuals is an intensive task for even a powerful cluster of computers. To this end, EMADE was designed to increase the size of the datasets used as an individual passes through stages. A program can first be tested with a small set of data; all the individuals that meet a threshold requirement are analyzed, and the top tier (strong Pareto strength) are then sent on to the next largest dataset.

Datasets are tracked by an `Age` attribute assigned to each individual. In this metaphor, age represents the knowledge and experience an individual acquires through its life. Throughout the evolution, this age increases as the individual evaluates each tier of data.

Individuals move from dataset to dataset as follows. `Age` starts at zero, indicating the individual needs to be evaluated on the first dataset. Evaluation begins on the smallest

dataset. When the objective scores are returned, EMADE adjusts them so that any individual that evaluates on a more mature dataset will dominate any individual that evaluated on an earlier dataset. For example, take the case of a minimization problem with a three-tiered dataset. If the two objectives are each bounded on $[0, 1)$, then we can choose an offset of 1, so that dataset 1 will yield adjusted scores of $[2, 3)$, dataset 2 will be bounded on $[1, 2)$, and dataset 3 will be bounded on $[0, 1)$. Figure 3.7 illustrates an example of the objective values for a similar three-tiered dataset. Once adjusted objectives are returned, a subpopulation N is selected for transitioning to the next level. Then, using thresholds, the subpopulation is divided into two pieces, specialists (which are below the threshold in one objective) and well-rounded individuals (which are below the thresholds in multiple objectives). EMADE selects $N/4$ specialists and $3N/4$ well-rounded individuals. The specialists help retain genetic diversity near the boundaries of performance. Each piece contributes a portion of the subpopulation using a SPEA2 sort and truncation selection.

After the individuals are selected, their age is incremented by 0.5 to indicate they have completed the prior dataset and are awaiting results on the next tier of data. When the individual returns from its maturation, its `Age` will be incremented by the remaining 0.5. The number of individuals N that are selected for advancement each generation degrades by a factor of two:

$$N = \frac{N_L}{2 \cdot (D_i + 1)},$$

where N_L is the launch size for new individuals, and D_i is the zero-indexed dataset number. Any individual that fails its evaluation on dataset 0 will not be selected to advance to the next level, meaning N only actually serves as an upper bound for the number of individuals selected to advance. This number is lower bounded by the number of individuals with valid fitness at age 1. In our three-tiered problem, if $N_L = 512$, then every generation 512 new individuals are created, 256 individuals advance from dataset 0 to 1, and 128 individuals advance from 1 to 2. This means a maximum of 896 individuals are placed into the evaluation queue.

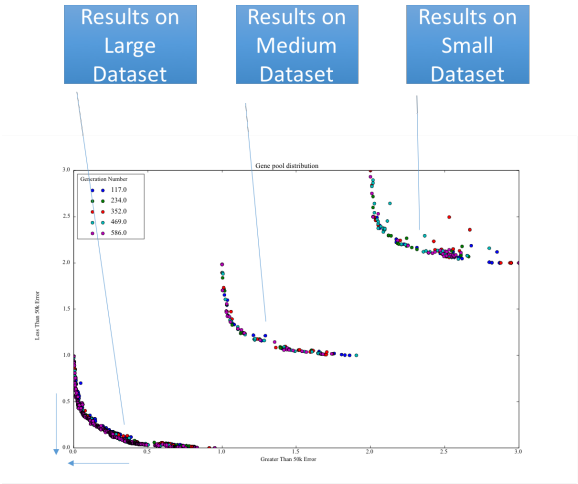


Figure 3.7: An example of a three-tiered structure where the objective scores for each dataset are bounded on $[0, 1)$ and being minimized.

This concept follows the idea of extra-genetic information in nature, knowledge acquired over the life span of a particular species. Tiered datasets in EMADE were inspired while solving the adult dataset problem from the UCI Machine Learning repository [28]. This problem presents the challenge of having 48,842 instances of data. Each instance comprises fourteen features: age, workclass, fnlwgt[1], education[2], education-num[3], marital-status, occupation, relationship, race, sex, capital-gain, capital-loss, hours-per-week, and native-country. With a dataset of this size, partitioning into a small and full dataset allowed individuals to "fail fast." More computer processing time was spent on individuals that

[1] Description of fnlwgt (final weight) from the dataset:

The weights on the CPS files are controlled to independent estimates of the civilian noninstitutional population of the US. These are prepared monthly for us by Population Division here at the Census Bureau. We use 3 sets of controls. These are:

1. A single cell estimate of the population 16+ for each state.
2. Controls for Hispanic Origin by age and sex.
3. Controls by Race, age and sex.

We use all three sets of controls in our weighting program and "rake" through them 6 times so that by the end we come back to all the controls we used.

The term estimate refers to population totals derived from CPS by creating "weighted tallies" of any specified socio-economic characteristics of the population.

People with similar demographic characteristics should have similar weights. There is one important caveat to remember about this statement. That is that since the CPS sample is actually a collection of 51 state samples, each with its own probability of selection, the statement only applies within state.

[2] Education is the highest level of education completed.

[3] Education-num is the number of years of education.

had already been somewhat proven, rather than on fatal alleles (genotypes that resulted in poor performance). This is similar to the work by Rohling [18] on computing expensive evaluations independently. However, rather than thresholding on time-consuming objective functions, we threshold on time-consuming and memory-intensive datasets.

In practice, on the adult dataset problem, out of 30,181 individuals, 24,294 were not necessary to advance to the larger datasets. At an average computation time of 258 seconds for an evaluation on the larger dataset, this represents a savings of over 1741 hours of computation time. By contrast, on the smallest dataset, only one percent of the total number of instances was used, and computation times averaged only 1.35 seconds. This means that our gain of approximately 1741 hours came at the expense of only 2.21 computational hours on the redundant computations required for those 5887 individuals that advanced. As a percentage, these times translate to accomplishing the same amount of individuals in 19.6% of the evaluation time, a savings of 80.4%. Chapter 4 shows more examples of tiered datasets saving significant processing time on other problem domains, including time-series and image-processing applications. Evaluation times and memory requirements tend to be significantly more demanding with these larger data problems, and the tiered dataset approach affords even more benefits in these domains.

3.11 Decentralized Control and Database

As EMADE moves from cluster configurations to cloud-based implementations, the limitations of SCOOP become apparent. For example, bringing new workers online after an optimization has started is not possible with SCOOP. To address the limitations of SCOOP, we added an option for database-driven control. With this control scheme, EMADE leverages a central database to support a scalable configuration, where a master and a set of workers can be brought up or down at any time so long as the database persists. The database consists of three tables, a history of all individuals that have been run in the evolutionary process, a set of non-dominated individuals at each generation, and a table showing

the status and performance of each individual.

The master process is responsible for maintaining the population of individuals and producing offspring from those it sends for evaluation. The master is also responsible for updating the non-dominated table each generation. The worker processes reads unevaluated individuals from the database, evaluates them, and then updates them for later use by the master.

Each individual in the database is referenced by a hash of the string representation of the parse tree. The hash is created when the individual is created and set as an attribute on the object. Therefore, the hash is only computed once over the lifespan of the individual, be it on the worker or master process.

Other columns in the individuals table include the pickled representation of the individual, each objective score that was computed on the individual, the ID of the last ordered dataset that the individual was evaluated against, the generation in which the individual first appeared, and the time to evaluate the most recent dataset.

When EMADE begins running, it can either start from an existing database, which we call "reuse," or wipe the database clean, which we refer to as "no-reuse." By running with reuse, we can have an effectively stateless optimization, as the full history of individuals is available at any time in the database.

The database can also be used to implement island model evolution and migrations through the use of multiple databases. Island models of evolution are common in the genetic algorithm domain in which subpopulations are evolved independently, with the exception of periodic migrations between subpopulations. Optimizations can be run concurrently or piecemeal, and individuals can be transferred from one database to the next to support the migration operation. While we do not explore the utility of these approaches in this research, the capability to study the impacts of this evolutionary strategy exists within our database structure.

The database can also be accessed outside the optimization to support analysis, extrac-

tion, and seeding operations. To seed the optimization, individuals can be added to the database manually either before the optimization, in which case it will appear in the initial population, or during the optimization, where it will be evaluated in the order it was inserted.

3.12 Batch Processing

As data size increases, the ability to keep entire datasets in memory simultaneously becomes more challenging. For applications with large data, we typically use batch processing to read and operate on data piecemeal. While not all methods can be implemented in batch mode, both the framework and those methods that can be implemented in batch are. Batch processing is also known as out-of-core processing, and machine learning methods that support it are often referred to as online learning methods. Some of these methods are sensitive to the size of each batch, while others are not. The specific methods that EMADE currently implements as primitives are the stochastic gradient descent (SGD) and passive aggressive methods from scikit-learn. When processing batched data in EMADE , the batch size is currently set as a hyper-parameter in the input document. Batch size is not included in the search space for optimization.

3.13 Summary

This chapter highlighted the unique capabilities of the EMADE infrastructure that set it apart from traditional genetic programming approaches and other autoML frameworks. The presented contributions allow for the application of automated approaches to challenging problems. In the past, it was not be possible to achieve human-competitive results using genetic approaches due to the complexity of the algorithms that using simple primitives could produce, the limited number of evaluations that the evolution could achieve with more complex primitives, and the lack of evolvability of past representations.

Next, Chapter 4 shows how the contributions from this chapter support a wide variety

of application problems, including those that were previously excluded from the domain of autoML. Perhaps more importantly, Chapter 4 shows that EMADE produces human-competitive solutions to these problems. Chapter 5 describes new approaches to aspects of the evolutionary process that the contributions from this chapter enable.

CHAPTER 4
DEMONSTRATIONS OF EMADE

This chapter demonstrates the capabilities of EMADE by applying it to several different types of problems, as shown in Table 1.1. Each problem type results in EMADE-produced algorithms competitive with human-made state-of-the-art in each domain. For the adult dataset in Section 4.1 and the Titanic dataset in Section 4.5, we compare our results with TPOT, an alternative autoML solution.

4.1 Feature Classification: Adult Dataset

The adult dataset can be found on the UCI Machine Learning Repository [28]. It comprises 48,842 points of data from the 1994 census, each with 14 features. The dataset consists of two pieces, a training set of 32,561 instances and a withheld validation set of 16,281 instances. The task for this dataset is a binary classification problem: given the 14 features, predict whether each individual made more than, or less than or equal to 50 thousand dollars. The features are age, workclass, fnlwgt, education, education-num, marital-status, occupation, relationship, race, sex, capital-gain, capital-loss, hours-per-week, and native-country. One of the challenges of this dataset is that the classes are unbalanced, i.e. there is a 76/24 split for individuals making less than 50 thousand and individuals making more, respectively. By producing a classifier which always chooses less than or equal to 50 thousand, you instantly have 76% accuracy in terms of overall error. In general, classification algorithms will favor the more common class. While simultaneously optimizing against multiple objectives allows for a full exploration of the trade-off space, regardless of the imbalance, the underlying favoritism of the classifiers to reducing false positives often results in an imbalance on the distribution of non-dominated solutions.

To prepare the dataset for EMADE, we first take the 32,561 points of training data

and run them through a vectorization process using scikit's dictionary vectorizer method. Vectorization transforms the 14 feature columns of categorical features, such as education, marital-status, and native-country, into multiple binary columns. In this expanded representation, each binary column indicates the presence of a particular categorical value. Each vectorized column set always has a single element set to true. The result of this process expands the original 14 features to 110 vectorized features.

Next, we cross-fold the training data into five training/testing pairs. Each pair comprises 80% training data and 20% testing data. A small sample of the the first fold forms our first tier dataset of 262 training samples and 69 testing samples.

To evaluate each individual algorithm, we use three objectives. The first two are apt for a binary classification problem: false positive rate, i.e. predicting a person made over 50 thousand dollars when they made less,

$$f_1(\hat{y}, y) = \frac{1}{\sum_{k=0}^{N} 1 - y_k} \sum_{k=0}^{N} |(1 - y_k)(\hat{y}_k - y_k)|,$$

and false negative rate, i.e. predicting a person made less than 50 thousand dollars when they made more,

$$f_2(\hat{y}, y) = \frac{1}{\sum_{k=0}^{N} y_k} \sum_{k=0}^{N} |y_k(\hat{y}_k - y_k)|.$$

For each of these objective functions, y_k is the truth value for each data point and $\hat{y}_k$ is the prediction made by the algorithm being evaluated. The remaining objective is the number of elements in the genome of the individual. Section 3.6 noted this objective rewards parsimony. We use a parsimony objective function in all subsequent applications of EMADE.

Because this section deals with pure feature data (which other AutoML platforms can handle as well), it makes sense to compare our results to results in literature. We compare EMADE against another DEAP-based AutoML framework, TPOT, as both TPOT and EMADE utilize evolutionary algorithms to evolve classifiers. Here, we run both EMADE and TPOT with population sizes of 512 individuals and 100 generations. This should rep-

resent on the order of 50,000 individuals evaluated. We use the same training data to cross-fold into five training and testing pairs for designing and optimizing individuals in both frameworks. We evaluate the Pareto-optimal algorithms from each framework on the same withheld data from the UCI repository, and plot the performance of each algorithm against false positives and false negatives.

The objectives define a Pareto front. Figure 4.8 show the frontier at the end of generation 100 for EMADE. We compute this frontier from the projection of the three-dimensional front onto the plane of objectives f_1 and f_2, i.e. false positives and false negatives, respectively. Every standalone machine learner is dominated by a chromosome created by the framework. Figure 4.1 shows the hypervolume, or interchangeably, area under the curve (AUC), as we are only considering two dimensions, of the non-dominated individuals on objectives f_1 and f_2 during the 100 generations of the optimization. We compute AUC using the left-hand Riemann sum of the objective scores of the non-dominated individuals as

$$AUC = \sum_{i=1}^{N-2} (x[i+1] - x[i])(y[i]), \tag{4.1}$$

where i is the ith Pareto coordinate pair when sorted by x coordinates in ascending order. For our problem, x represents false positive rate (f_1) and y represents false negative rate (f_2). During the evolutionary process, there are long periods of stagnation with little to no change occurring between generations followed by abrupt reductions. This punctuated equilibrium phenomena is something commonly observed in nature as well, where a population is relatively stable until a useful mutation is discovered and quickly spreads through the population as it reaches its next equilibrium.

Figure 4.2 shows the number of non-dominated individuals in each generation over the course of the optimization. The general increase in the number of non-dominated individuals over time indicates that the genetic algorithm is successfully exploring the trade-off space between objectives. The small drops along the way indicate that the search produced an individual that dominated several others in the non-dominated frontier. This may or

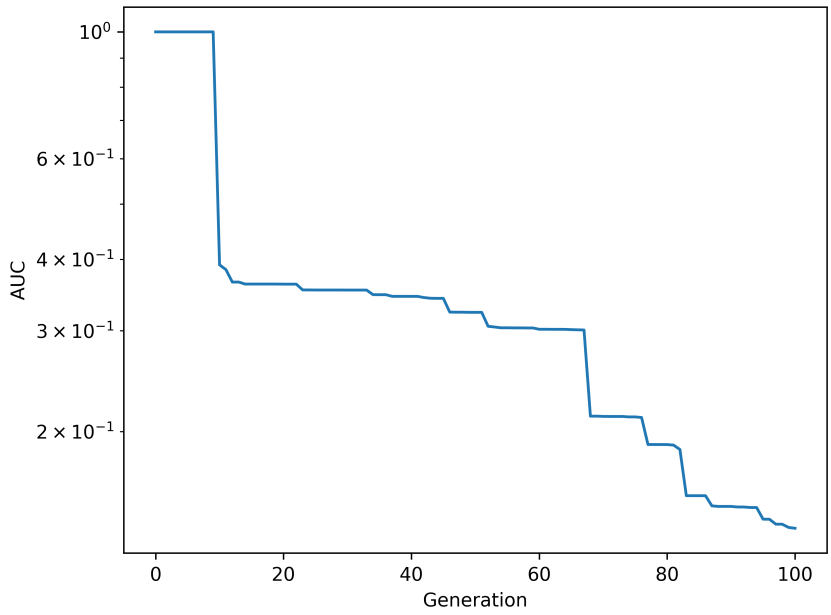


Figure 4.1: Area under the curve enclosed by non-dominated individuals on objectives f_1 and f_2 (i.e., false positives and false negatives) during adult dataset optimization.

may not coincide with a drop in area under the curve, depending on which objectives the new individual dominated the old individuals. It could be the case that the new individual achieved the same results in f_1 and f_2, but did so in fewer elements, and so the AUC does not change.

Figure 4.3 shows a box and whisker plot of the time it takes to evaluate each individual in the population by generation. The orange line traces the median time in each generation, while the green line traces the mean. Each generation, the box spans the first (Q1) to the third (Q3) quartile of the evaluation times, from which the difference defines the interquartile range ($IQR = Q3 - Q1$). The whiskers extend from $Q1 - 1.5 * IQR$ to $Q3 + 1.5 * IQR$. Any evaluation times outside the range $[Q1 - 1.5 * IQR, Q3 + 1.5 * IQR]$ are considered. The blue dots on the figure show the evaluation times of these outliers.

Despite the emphasis placed on simplicity of the algorithms, on average they grow more computationally expensive as time goes on. There is a hard cutoff at 9000 seconds for an evaluation. This cutoff is a parameter set in EMADE, which we can see is set appropriately,

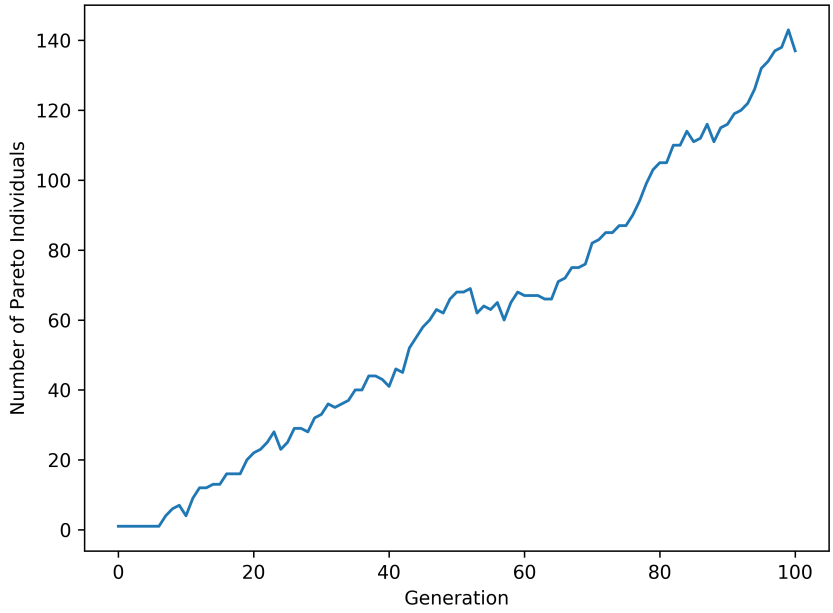


Figure 4.2: Number of non-dominated individuals in each generation.

since all individuals that are stopped because of processing time are outliers. This cutoff has an analogy to nature that is to the amount of time it takes to produce offspring. Nature bounds the time when an organism is able to produce viable offspring. This upper bound stops the evolutionary process from spending to long to produce solutions, enabling faster generations. This is an advantage in nature as shorter times between generations allow for more adaptable populations.

In the beginning of the optimization (i.e., generations 0 through 18), the average evaluation time is drastically lower, since the evolution has not yet discovered an algorithm that can make it past the first tier dataset. The average evaluation in generation 100 completed significantly faster than the generations that proceeded it. This anomaly comes from the fact that the experiment ended at the beginning of generation 101. In a traditional, synchronous evolution, the beginning of generation 101 would mean that every individual in generation 100 completed their evaluation. However, EMADE uses an asynchronous evolution technique. We use this technique to ensure that EMADE is always evaluating individuals in parallel, rather than having many of idle workers waiting for the laggards of

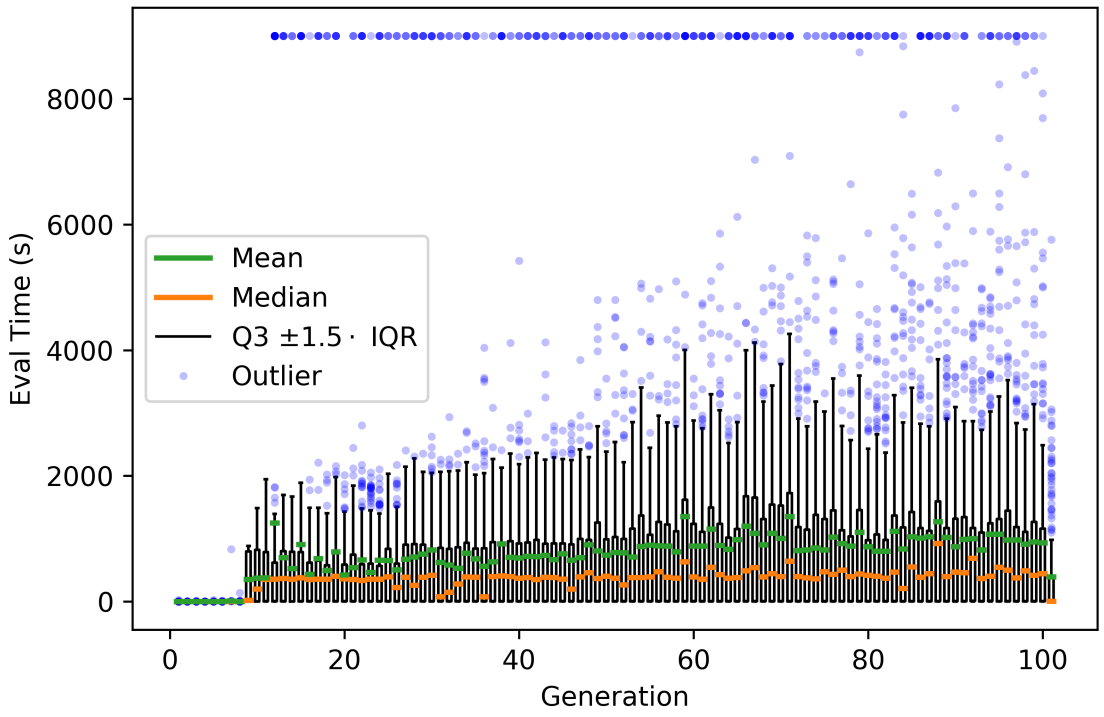


Figure 4.3: Box and whisker plot of the amount of time it takes to evaluate one individual in the population by generation.

one generation to finish. Because of our asynchronous evaluation process, generation 101 beginning, does not guarantee that every individual in generation 100 completed its full evaluation. We also know that it takes multiple generations for individuals to move from tier to tier when using multiple datasets. By the time we stopped EMADE in generation 101, 50 individuals completed evaluation on the full dataset from generation 100, while 182 individuals were only evaluated on the first tier dataset. Contrasting these numbers with individuals from generation 99, 98 individuals were evaluated on the full dataset while 88 were only evaluated on the first tier.

Figure 4.4 shows the time between each generation during the evolution, i.e. from the start of generation k to the start of generation $k + 1$. The amount of time between generations increases throughout the evolutionary process. We expect this increase in inter-generational time to increase because we know that average evaluation times per individual increases throughout the evolutionary process as well. The inter-generational time varies,

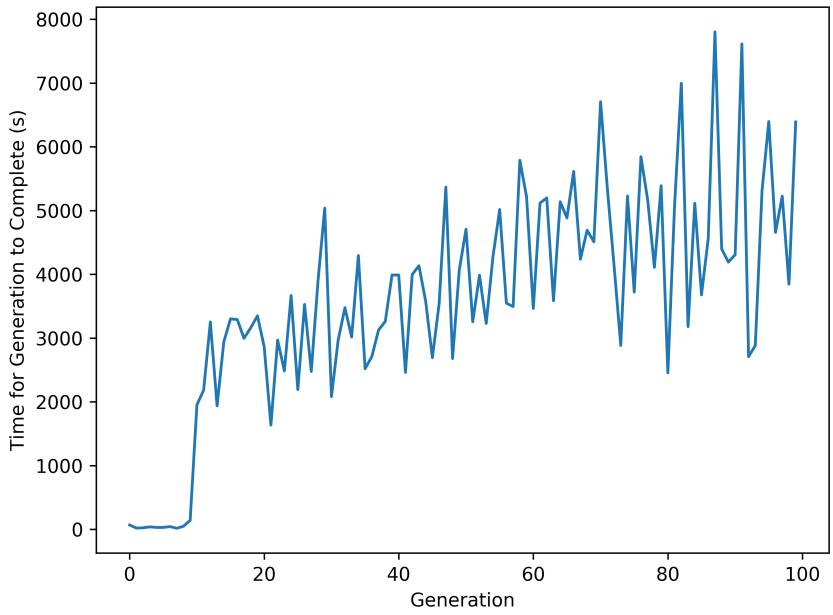


Figure 4.4: Inter-generational time throughout the course of an evolution.

despite the clear trend upwards trend. Neighboring generations can vary by as much as 100% toward the end of the experiment. This variability could be in part due to system loads, but may also just be a natural by-product of the algorithms that each generation produces.

Figure 4.5 shows the individual with the best overall accuracy of 87.64% on the 5 cross-folds run through EMADE. This accuracy is characterized by a false positive rate of 5.5% and a false negative rate of 34.2%. This follows our intuition; with the imbalance of the dataset, we achieve the best accuracy mainly by suppressing false positives, as 76% of the dataset is made up of negative examples, i.e. those who have made less than 50 thousand dollars.

The algorithm in Figure 4.5 has quite a bit of genetic bloat. The root of the tree is a `SelKBest`, which is a feature reduction method. This feature selection method makes no alterations to the classifications of the algorithm and thus has no impact on performance. Figure 4.6 shows this same genome with the feature reduction method removed.

Once the bloat is removed, the genome is still more complex than the algorithm it

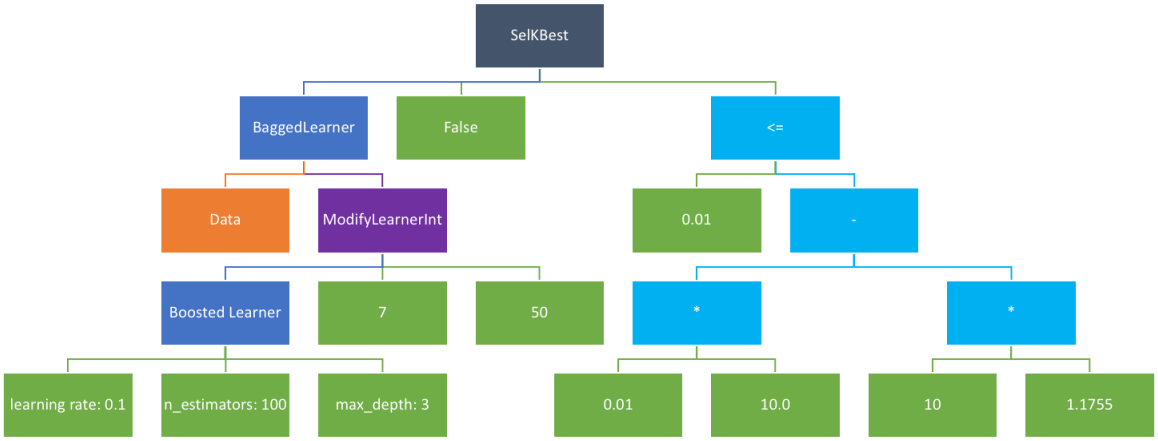


Figure 4.5: Individual with best overall accuracy scored on the cross-folded training and testing data from the vectorized adult dataset.

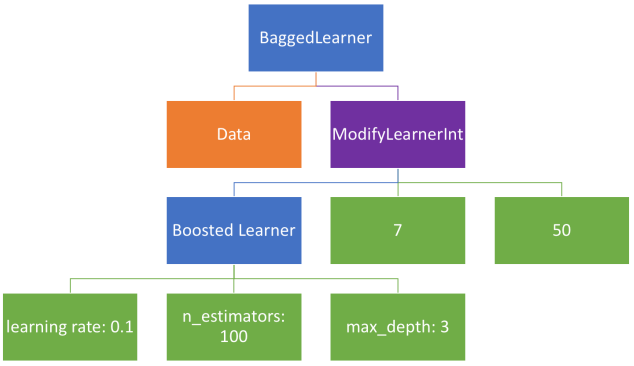


Figure 4.6: Active portion of genome from Figure 4.5.

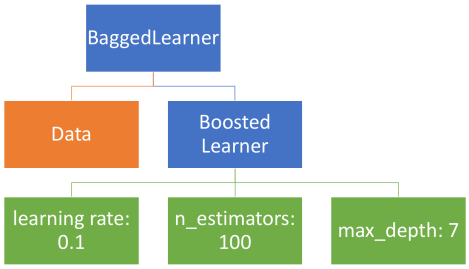


Figure 4.7: Effective algorithm after simplification from Figure 4.6.

represents. The active genome in Figure 4.6 contains a `ModifyLearnerInt` method that effectively changes the maximum depth of the `Boosted Learner` from 3 to 7. Figure 4.7 shows the simplified genome.

The final algorithm is simply a combination of two learning techniques. The bagged learner consumes the boosted learner as an input and uses it to fit multiple random subsets of the data. The bagged learner then aggregates the classifications of the multiple boosted learners to create the final classification of the algorithm. The underlying boosted learner uses an AdaBoost algorithm to assign more importance to misclassified samples during training.

After running the evolution for 100 generations, we chose the final non-dominated set of algorithms produced by EMADE for evaluation against the validation set. The validation set contains 16,281 points of data. We put these points of data through the same pre-processing steps applied to the training data. Figure 4.8 highlights the similarity between the performance of EMADE's algorithms on the cross-folded testing data and the withheld validation data. On the other hand, the algorithm produced by TPOT, which had a testing accuracy of 87.57%, validated to an accuracy of 87.13%, while EMADE validated to an accuracy 87.61%. EMADE's ability to resist overtraining is due to its emphasis on multiple objectives. While TPOT produced only ten solutions on two objectives of accuracy and complexity (number of elements), EMADE used three objectives to produce 137 algorithms. Figure 4.9 shows a zoomed-in view of the non-dominated frontier to highlight the difference in final solutions produced by EMADE and TPOT.

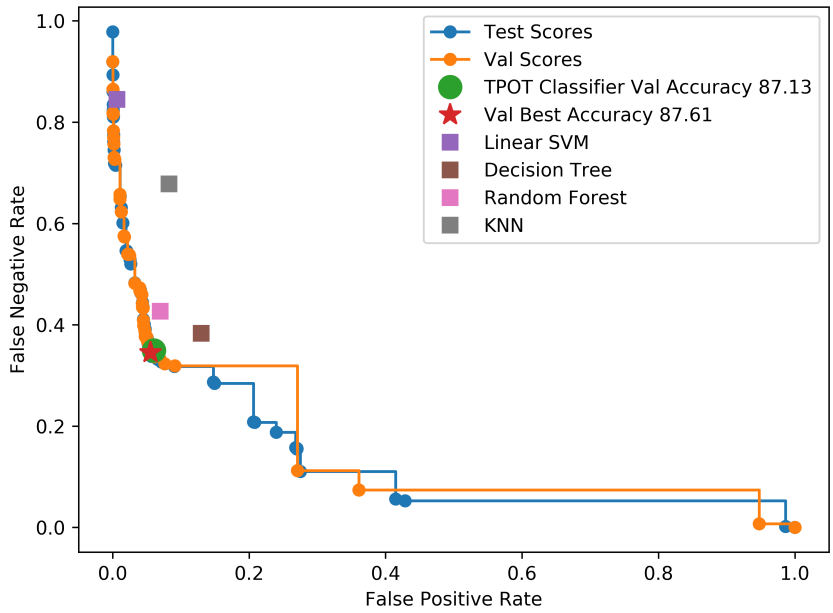


Figure 4.8: Non-dominated frontier on objectives of false positive rate and false negative rate.

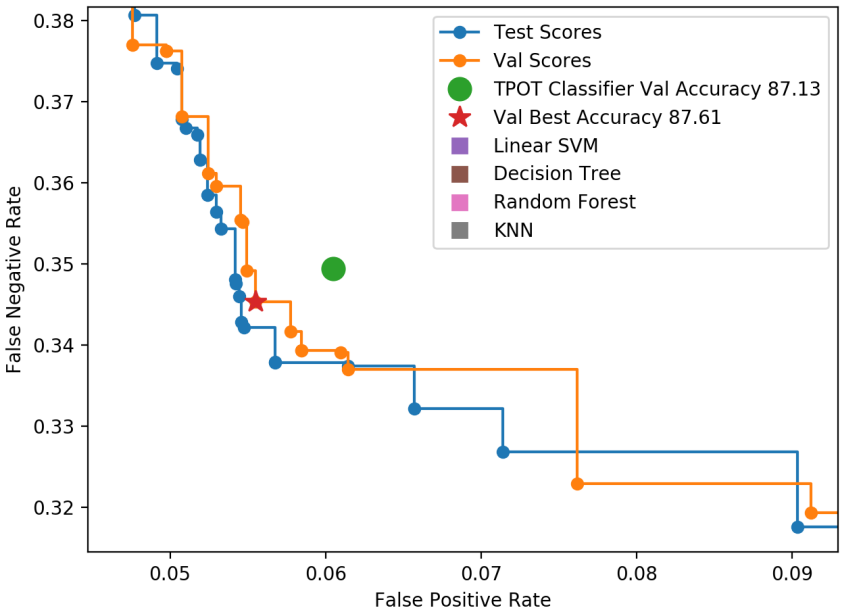


Figure 4.9: Non-dominated frontier on objectives of false positive rate and false negative rate, zoomed-in to show comparison with TPOT.

4.2 Time Series Classification: AGLogica

AGLogica is a company interested in behavior classification of companion animals to support veterinary care. They aim to produce objective measurements of behaviors that correlate with medical issues to better monitor the results of care and treatment plans. EMADE produced high-quality classification algorithms that detect various behaviors from accelerometer data, such as scratching and shaking. AGLogica uses both of these behavioral algorithms in their Vetrax cloud platform to identify *pruritic* conditions and monitor their treatment. Pruritic conditions are those that cause animals to itch or scratch.

4.2.1 Data Collection

Because EMADE requires supervised data, and no available canine accelerometer data set exists, AGLogica collected a robust data set from two Atlanta Humane Society (AHS) locations, as well as the Animal Dermatology Clinic (ADC). In total, data was captured from 573 dogs: 235 from the AHS Howell Mill campus, 250 from the AHS Mansell campus, and 88 from the ADC.

For each data dog in the study, AGLogica collected both video and sensor data. The video was captured using various digital platforms, including a GoPro Hero4, Nexus 7 tablet, and Cannon VIXIA HF R600. The accelerometer data was captured from an Axivity AX3 sensor sampling at 100 Hz. AGlogica used an annotation tool from the Language Archive called ELAN (the EUDICO Linguistic Annotator) to label actions that occurred during each data collection. Annotating the data began with a synchronization of video and accelerometer information. The annotator aligned the two data sources based on a strong up/down motion performed as the accelerometer was held in front of the camera at the beginning of each collection. Then, with accelerometer data hidden, each video was hand annotated by two observers using the controlled vocabulary shown in Table 4.1. The common annotations from the two observers were then exported to line up with the

Table 4.1: Controlled Vocabulary for Behavior Annotation

Gait	Activity	Scratching	Licking	Interaction	Ignore
Walking - Off Leash	Shaking	Front Leg - Body	Licking Paws	Petting	Not In View
Walking - On Leash	Drinking	Front Leg - Head	Licking Body		Other
Running - Off Leash	Eating	Hind Leg - Body	Other		
Running - On Leash	Chewing	Hind Leg - Head			
Laying - Resting	Barking	Other			
Sitting - Resting	Excreting				
Standing - Resting	Urinating				
	Digging				

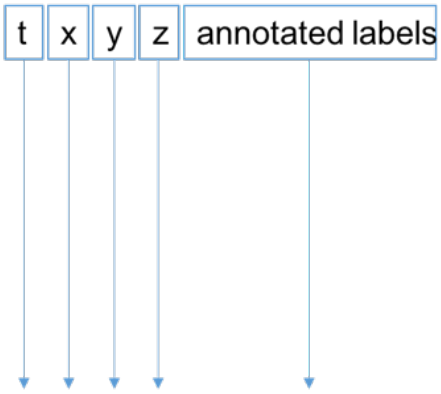


Figure 4.10: Data is organized in rows when exported from ELAN, where each row represents one sample of data taken by the accelerometer.

accelerometer data such that every sample from the sensor was labeled with the activity.

Figure 4.10 shows an example of how an exported annotation is formatted. The columns t, x, y, and z represent the readings from the accelerometer, t being time, and x, y, z being the three axes of the accelerometer at each time. The annotated label is produced from the annotations in ELAN.

4.2.2 Data Processing

Once the data has been exported from ELAN, each dog has its own file with the columns time, x acceleration, y acceleration, z acceleration, and annotation. We drop all unannotated rows from each file, and then break the remaining rows into one second frames of data. We label each frame from their associated annotations. Each labeled frame is essentially a 3x100 matrix, where the three rows are the three accelerometer axes and the 100 columns contain the 100 Hz samples for each labeled second. We also label each frame with the file

it came from, ensuring that the data can be folded in manner where no data from a single dog can appear in more than one set.

Because AGLogica performed their data collections for relatively short and carefully observed periods of time, most of our data comes from regularly positioned accelerometers. The position in question sits directly below a dog's neck, with the z axis capturing up/down motion, the x axis capturing left/right motion, and the y axis capturing forward/backward motion. Because all our data is captured from a similar location, any algorithms produced from this data will be biased to the same location. To better leverage these algorithms in real-world scenarios, we transform each 3x100 frame by multiplying it with a rotation matrix of a random angle about the y-axis:

$$\text{frame}_{\text{corrected}} = R_y(\theta) \cdot \text{frame}_{\text{raw}} = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \cdot \text{frame}_{\text{raw}}, \tag{4.2}$$

where θ is randomly generated for each frame that we transform. Rotating the data around random angles about the neck (y-axis) helps add robustness to the dataset to prevent over-training.

For scratching and shaking behaviors, we filter out data where the coefficient of variation (CoV= $\frac{\sigma}{\mu}$) of the frame is less than 0.2. Because both of these are high-energy behaviors, we prune down the data where little to no activity is occurring to help suppress the false positive rates, as well as correct any over-annotations that occurred during the hand annotation process. Over-annotations occurred when observers marked longer windows of time in the video than the behavior took place. Removal of low-energy frames ensures we are only making high-energy classifications on high-energy measurements.

We determined the threshold of 0.2 empirically by analyzing the training and testing sets visually. We started with a low CoV threshold, and slowly raised it while inspecting each frame of data of "scratching" and "shaking." We looked for a high enough CoV

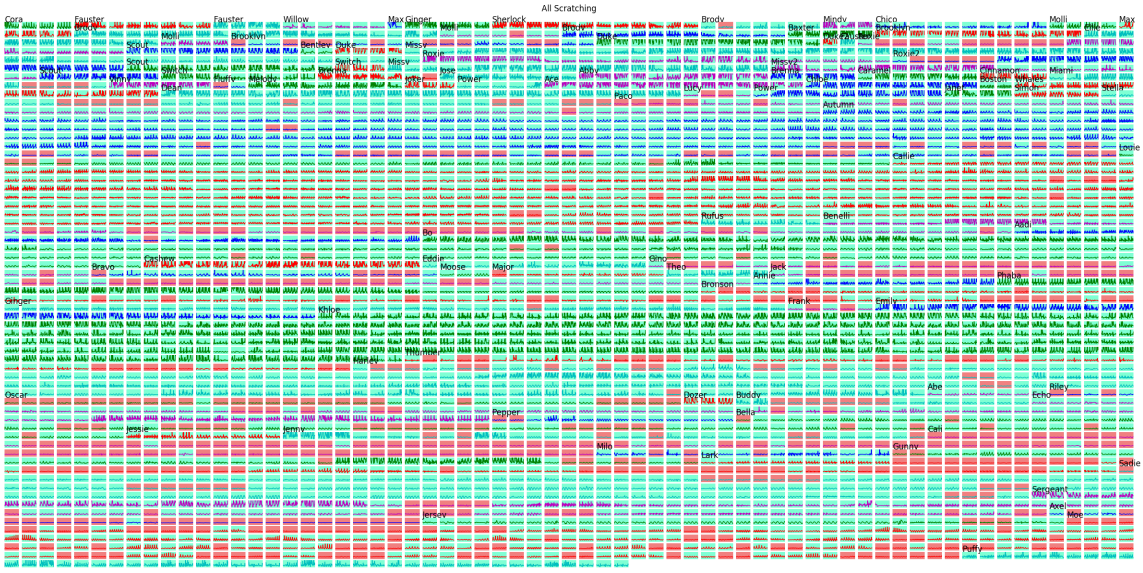


Figure 4.11: A grid of all scratching data in the training set. Boxes with a green background have a CoV > 0.2, while those with a red background have a CoV ≤ 0.2. A change in color of the line indicates a change in dog from which the frame was taken.

to remove frames where virtually no activity was present on the accelerometer, without removing any frames that had activity. Figure 4.11 is an example visualization of the CoV process for scratching data, where the frames shown with a green background are marked as passing the CoV test, and those with a red background are marked as failing the CoV test.

For the evolutionary machine learning process, we organize our data into two chunks: the first is the set of data that we use to train and score the models, the second we withhold until the final algorithms have been chosen and then use it to validate the algorithms on data to which they have not been previously exposed. We cross-fold the first set of data into five 80/20 training/testing splits. The training data is a random selection of 80% of dogs in the first set of data; we use it to fit machine learning models in the tree structure. The testing data is the remaining 20% of dogs in the first set of data; we use it to compute objective scores and assess performance. We average the objective scores over the five splits of the data to get some sense of robustness for the algorithm. When finally implemented for validation, we use the combined set of training and testing data for training the machine

learning methods.

We took a small redundant portion of one of the folds to be a lower tier data set. This enables EMADE to quickly identify algorithms that are unable to perform successful classifications without requiring the processing of the full dataset.

4.2.3 Objectives

While the overall goal of the Vetrax platform is to perform multi-class detection, we implemented this as a series of binary classifiers in a rejection chain, as shown in Figure 4.12. For each classifier, the objectives of false positives and false negatives are necessary. Both the shaking and scratching events being observed are rare in the data set, making up about 4% of all one-second frames. This unbalanced data set means that high accuracy can easily be achieved, but having strong confidence in the predictions by the classifier is more difficult, due to the metric of precision (also called positive predictive value). Precision is the ratio of true positives to all positives reported by the algorithm (i.e. true positives and false positives). For example, a 4% false positive rate and 0% false negative rate would lead to a precision value of just 50%; when we see a prediction from the classifier, we may as well flip a coin as to whether or not the dog was truly performing the activity. While precision was not used as an objective in the optimization process (as it is just a weighted combination of false positives and false negatives), it was strongly considered when looking at thresholds and algorithm selection.

4.2.4 New Primitives

AGLogica reached out to an expert in canine behavior classification for implementation of their solution. One method they used was the empirical cumulative distribution function (ECDF) coefficient method [54], which we implemented in EMADE.

For ECDF, on each axis, accelerometer readings are sorted in ascending order, then N linearly spaced indices are sampled from the sorted array. The $3xN$ matrix of sample is

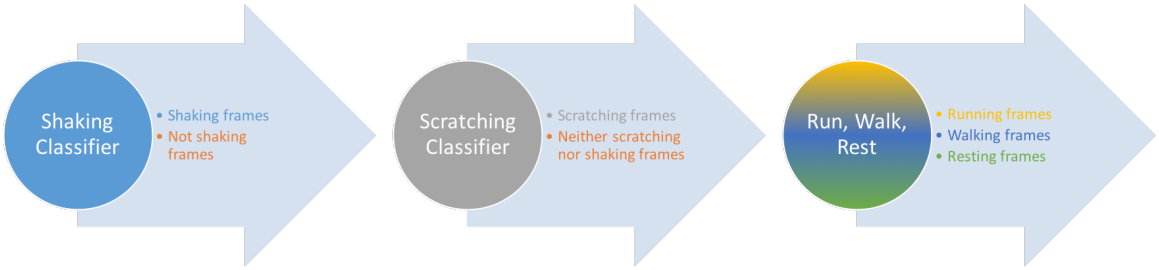


Figure 4.12: Example of a rejection chain of binary classifiers. Each of the shaking and scratching classifier only need to identify a single behavior. The frame of data will pass through the chain until it is tested positive by one of these classifiers. At the end a placeholder is in place to label the frame as running, walking or resting based on the total energy in the frame.

then flattened into a single dimensional array of length $3N$ and appended with the means of each of the axis for a total of $3N + 3$ samples. Interestingly, without seeding any pre-made tree configurations, EMADE found the ECDF coefficients to be a powerful primitive for working with the accelerometer data.

4.2.5 Results

Because EMADE-evolved behavioral classification algorithms are a market differentiator for AGLogica, we cannot share their implemented algorithms. However, we have permission to show developmental algorithms. Figure 4.13 shows a walking algorithm produced by EMADE. While the performance of this algorithm was not sufficient to be a non-dominated individual (and thus we are permitted to share it), it does demonstrate the ability of EMADE to work with multi-dimensional time-series data. For this algorithm, EMADE discovered the utility of the spectral representation of the accelerometer data combined with a high pass filter to isolate the higher frequency components of the signal. These higher frequency components help discriminate walking from other behaviors. The algo-

rithm next uses a feature selection method to reduce the feature space to 11% of its size prior to feeding the results to a triple-stacked learning procedure. The learning procedure combines a Gaussian mixture model that feeds its prediction, along with the feature set, to a bagged random forest classifier. Finally, the predictions of both the mixture model and the bagged random forest combine with the selected spectral features to train a Bayesian classifier.

This individual has a false positive rate of 31.33% and a false negative rate of 12.45%. The high false positive rate comes from confusion with running. This algorithm is significantly more accurate than the prior state-of-the-art of choosing energy-based thresholds and applying them to the particular sample. The energy calculation is

$$\sum_{i=0}^{99}(\sqrt{x[i]^2 + y[i]^2 + z[i]^2} - 1), \tag{4.3}$$

where i is the sample index in each 100 Hz frame of data; x, y, and z are the accelerometer values in g's of each axis, respectively. Figure 4.14 shows the analysis of true positive (i.e. 1 - false negative) and false positive performance based on selecting two thresholds to segment data between resting and walking, and walking and running. For comparison to the EMADE algorithms, we select thresholds as follows. We begin with the selection of a low threshold to lock in a true positive rate for resting. Reaching the 90th percentile requires a low threshold of at least five. Next, for comparing to EMADE, we choose a high threshold to match true positive rates for walking. With a low threshold of five, this 87.55% false positive rate requires a high threshold around 60. With this configuration, the energy calculation method has a false positive performance in the 50th percentile range. By these measures, the EMADE-produced algorithm reduced false positives by 37.3%, i.e. $\frac{orig-new}{orig}$. The energy calculation in Equation 4.3 can still be used for running and walking.

While we cannot share algorithm details, we can share some performance information from the development of the scratching and shaking algorithms. Figures 4.15 and 4.16

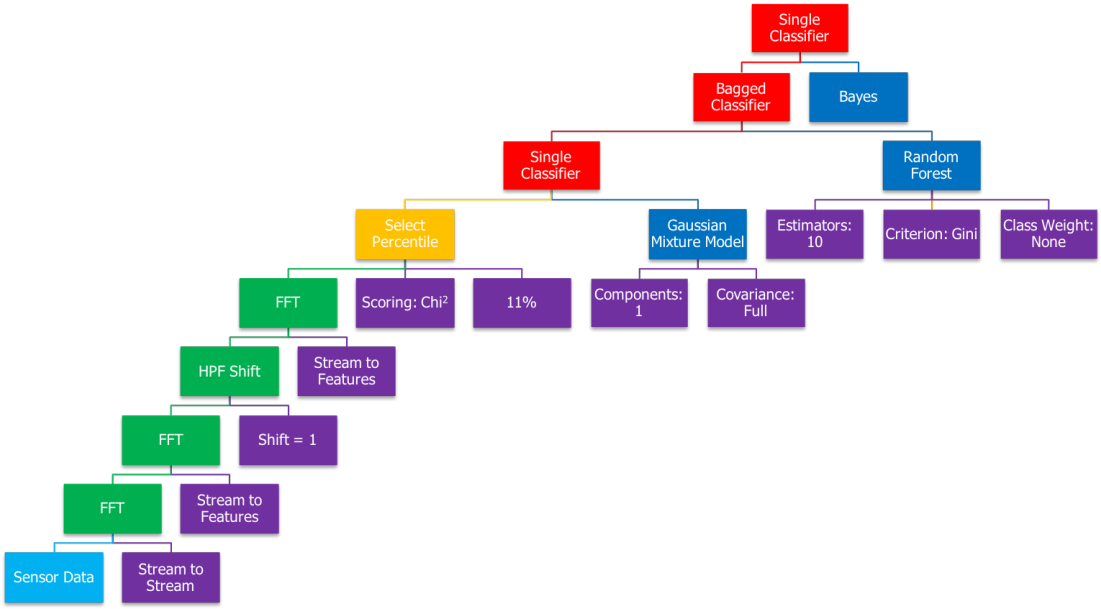


Figure 4.13: Example algorithm evolved by EMADE to classify walking behaviors.

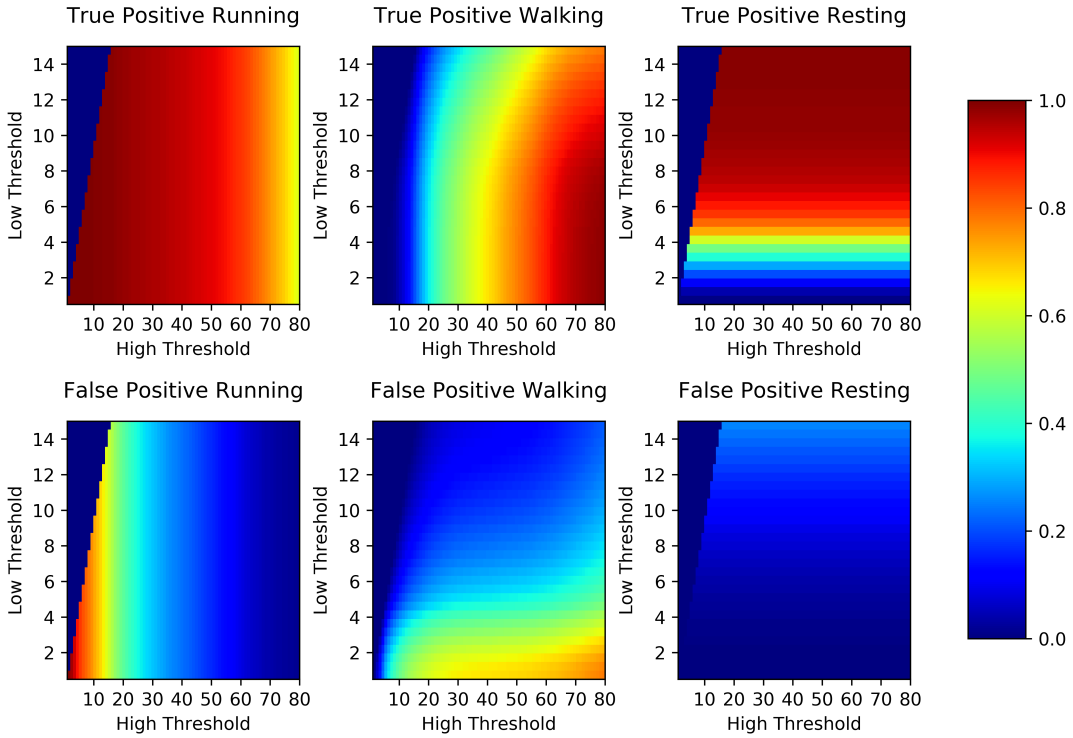


Figure 4.14: Performance for running, walking, and resting segmentation based on two thresholds on total energy.

show non-dominated frontiers for shaking and scratching algorithms, respectively, scored on testing data. The shaded areas represent the positive predictive value (PPV) for each region of objective space, based on the rate of occurrence of each behavior in the dataset.

After applying the testing data set, the head shaking algorithm selected for implementation showed sensitivity of 72.16% and specificity of 99.78%. In our testing, shaking made up 0.81% of the data, resulting in an overall accuracy of 99.56%, a positive predictive value of 72.57%, and a negative predictive value of 99.78%.

We validated the final scratching algorithm selected for implementation to show sensitivity rate of 76.85% and specificity rate of 99.73%. Scratching made up 2.12% of the data, resulting in an overall accuracy of 99.24%, a positive predictive value of 86.07%, and a negative predictive value of 99.50%.

The low false positive rates and high positive predictive values allow the shaking and scratching algorithms to produce reliable trends. Combined with a baseline measurement of a dog prior to an intervention, these trends can be used by a veterinarian to objectively understand how an animal responds to treatment.

4.2.6 Discussion

This section demonstrates the power of EMADE and autoML to generate extremely accurate classification algorithms through multi-objective optimization. This work also represents the first application of autoML through genetic programming to time-series data. Furthermore, the results featured in this section led to cloud-implemented algorithms, providing an industry-first objective tool for aiding management of pruritic conditions in dogs.

4.3 Feature Regression: Heat Stress Prediction

Emergency responders are required to wear personal protective equipment when responding to a variety of threats, primarily chemical, nuclear, and biological agents. In addition to the dangers posed by these substances, the personnel also risk suffering the effects of heat

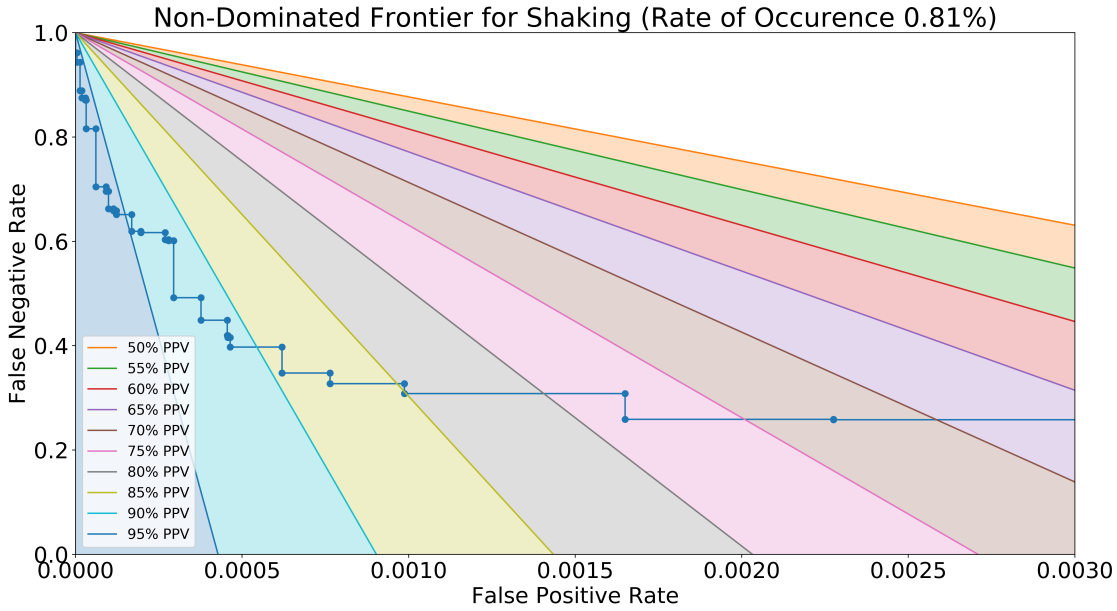


Figure 4.15: Non-dominated frontier of performance of shaking classifier on cross-folded test sets.

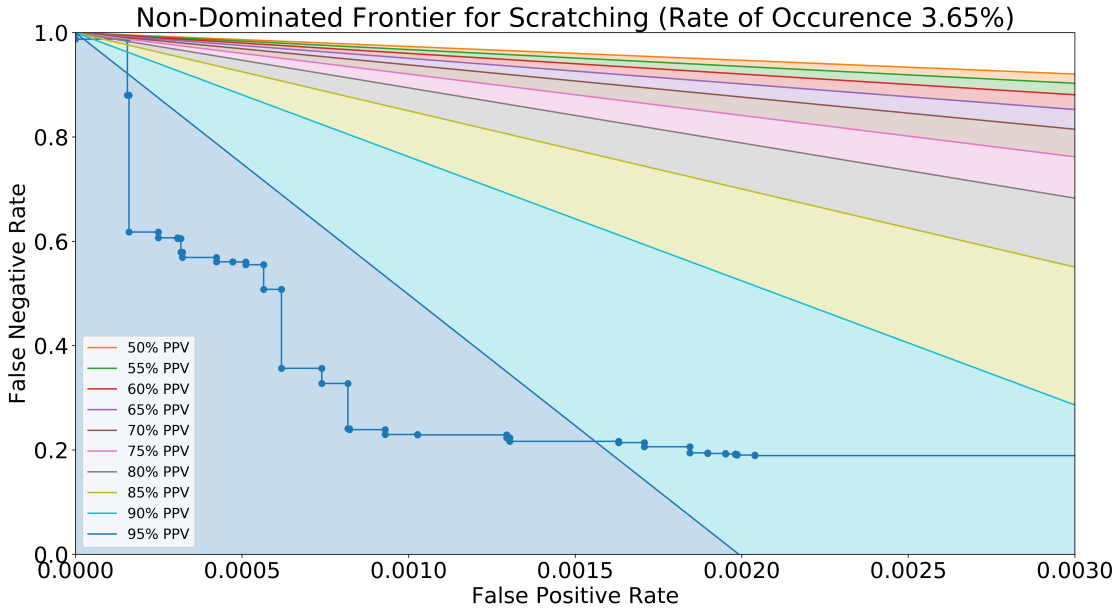


Figure 4.16: Non-dominated frontier of performance of scratching classifier on cross-folded test sets.

stress from the microclimate inside their hazmat suits, the type of work being performed, and the external environment.

The effects of heat stress can be detected in changes in a user's core temperature or heart rate.[1] If a subject's heart rate is sustained[2] above $0.9 \times (220 - \text{Age})$ or the subject's core temperature exceeds 38.5 degrees Celsius, then the emergency responder is considered to be at risk. It is not recommended that an operator remain within a vapor-tight protective garment for greater than one hour. Because of this guideline, we use an hour as an upper bound for our trials. The earliest time in the trial at which any of these constraints is violated is called the *expected pull-time*.

This section, uses EMADE to create algorithms that predict expected pull-time, given characteristics of the user, the type of suit, and the environment. This application showcases EMADE's ability to operate on feature data.

4.3.1 Data Collection

Data was collected at two facilities by The Netherlands Organisation for Applied Scientific Research (TNO) in Delft, Netherlands and North Carolina State University (NCSU) in Raleigh, NC, USA. Subjects were asked to use a treadmill at varying intensities in different environments, wearing a selection of suits. In total, 163 trials were performed. In each trial, numerous sensors recorded data: one heart rate sensor, one core temperature sensor, five microclimate relative humidity sensors, five microclimate temperature sensors, and five different skin temperature sensors. Depending on the trial, data was sampled from each of the sensors at a rate of one sample every ten or fifteen seconds.

[1] Throughout this section, heart rate is measured in beats per minute, temperature is measured in degrees Celsius, and age is measured in years.

[2] By sustained, we mean that the heart rate continuously stays above the threshold for at least a minute.

4.3.2 Physics-Based Model

We received a first-principles physics-based model from our sponsor. It was developed by physiologists for thermal analysis, and provides time series predictions for a subject's core temperature based on 331 parameters that describe the clothing worn, the activity being performed, the environment the work was performed in, and the physical attributes of the user. The model does not provide the ability to predict heart rate. Also, for accurate predictions, it requires information that will be unavailable to the user at time of use; most notably, an initial core temperature. The results of this model are used as one of our features, which are described in the next section. In the context of EMADE, using this model as a set of features is equivalent to forcing this node to be a primitive in every tree structure.

4.3.3 Feature Space

Although EMADE has the capability to work with time-domain data frames, for this problem we concentrated on operating directly in feature space. We framed the problem in this manner due to the unavailability of real-time heart rate and core temperature data from the subject during a trip downrange. All predictions will need to be constructed from actionable situational data: the physical description of the subject, the environmental parameters that can easily be measured in the field, and the parameters that make up the suit. The eight features we use to predict expected pull-time are age (years), height (meters), weight (kg), intensity of the work being performed (discrete value, indicating 2.5, 4, or 5.5 km/hr), temperature (°Celsius), relative humidity (percentage), suit (discrete value, meaning one of six hazmat suits), and the prediction made by the physics model. The physics model's predictions were all generated beforehand using the software package provided by TNO on each row in our data set, to increase the efficiency of running the optimization. In the field, they would be generated on demand.

Out of the 163 trials performed, 120 were used for constructing the feature set. The trials in which the activity was terminated prior to the expected pull-time were excluded,

because in these instances we do not have a truth value that we are able to compare to our predictions.[3]

4.3.4 Evaluation

Individual algorithms were evaluated using five randomly generated folds of the data set. Care was taken to ensure that if a subject appeared in multiple trials, then all of that subject's trials would be in either the training or the testing set. For each fold, 80% of the subjects were placed in the training set and the remaining trials were used as testing data. The objective scores are calculated based on the algorithm's ability to successfully predict values for the test data, and each objective was averaged over the five folds of the data.

Evaluation was performed on a cluster of eighteen Linux computers. The individuals were distributed through DEAP's built-in integration with SCOOP, which works in conjunction with the cluster's management software, Son of Grid Engine (SGE), to distribute the load across the nodes effectively. We used a parallel environment allocation rule in SGE to ensure that only three jobs were running on each machine. This yielded fifty-three total evaluation slots and one root worker performing the evolutionary loop.

4.3.5 Evolutionary Parameters

For this problem, we used four mating operators with different rates of application: one-point crossover (50%), ephemeral-only crossover (50%), headless chicken crossover (10%), and headless chicken ephemeral crossover (10%). The mutation methods used and their rates of application were: insertion (5%), ephemeral (25%), node replacement (5%), uniform (5%), and shrink (5%). We applied each operator independently across the population.

The elite pool was chosen to be a fixed size of 512 individuals. Two hundred individuals were selected to be evaluated in each asynchronous generation, meaning selection

[3] Actual pull-time is the time at which the trial was actually concluded. This can be less than the expected pull-time due to a measurement error or a subject voluntarily terminating his or her trial early. Both cases occurred.

was generally performed from approximately 712 individuals, depending on the arrival of results.[4]

4.3.6 Objective Functions

EMADE produced algorithms in two phases of optimization with two different sets of objectives.

Optimization 1

If $\{y_j\}_{j=1}^N$ are the expected pull-times (truth values) and $\{\hat{y}_j\}_{j=1}^N$ are the corresponding predictions (so that $\hat{y}_k$ is the predicted value of y_k for $k = 1, 2, \ldots, N$) and if $J = \{1, 2, \ldots, N\}$ is the set indexing the sequences, then $J_{\text{under}} = \{j \in J : \hat{y}_j < y_j\}$ is the set of indices where *under predictions* occurred and $J_{\text{over}} = \{j \in J : \hat{y}_j > y_j\}$ is the set of indices where *over predictions* occurred. In the context of the application of the models we create, an over prediction puts the first responder at risk, i.e. we predict they can work longer than they are able to safely. An under prediction puts the task the responder is trying to accomplish at risk, since we predict they have less time than they do to perform their critical task. There is a meaningful trade-off space here, and different situations certainly call for different tolerances of error.

The first set of objective functions were chosen to minimize different types of prediction errors. They were:

- Root Mean Squared (RMS) error, $\sqrt{\frac{1}{N}\sum_{k=1}^{N}(y_k - \hat{y}_k)^2}$
- mean over prediction, $f(J_{\text{over}})$
- mean under prediction, $f(J_{\text{under}})$
- percent error, $\frac{1}{N}\sum_{k=0}^{N}\frac{|y_k - \hat{y}_k|}{y_k}$

[4]EMADE does not use generation synchronization, but rather creates a queue of solutions to evaluate. It spawns a new generation when the queue falls below a certain size (in this case, one hundred individuals).

- the depth of the algorithm's tree

where $f(K) = \frac{1}{|K|} \sum_{k \in K} |y_k - \hat{y}_k|$.

Optimization 2

For the second cycle of optimization, the Pareto front from the first cycle was used to seed the optimization. This set of objective functions was designed to explicitly reduce the variability in error in addition to the size of the error. These functions are:

- mean over prediction
- mean under prediction
- count of over predicted trials, $|J_{\text{over}}|$
- count of under predicted trials, $|J_{\text{under}}|$
- standard deviation of over prediction, $g(J_{\text{over}})$
- standard deviation of under prediction, $g(J_{\text{under}})$
- maximum over prediction, $h(J_{\text{over}})$
- maximum under prediction, $h(J_{\text{under}})$
- the depth of the algorithm's tree

Here, $g(K) = \sqrt{\frac{1}{|K|} \sum_{k \in K} (x_k - \bar{x})^2}$ is the standard deviation of $x_k = |y_k - \hat{y}_k|$ over the set K and $h(K) = \max_{k \in K} |y_k - \hat{y}_k|$.

4.3.7 Results and Discussion

Optimization 1

Beginning our evolutionary process with a human-generated solution affords us a point in objective space against which we can compare our automatically generated algorithms. In

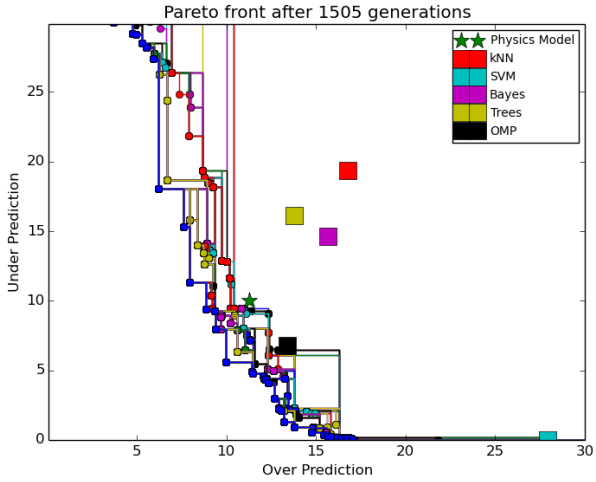


Figure 4.17: Pareto fronts of solutions generated by EMADE when projected from five dimensions to two dimensions: mean over prediction and mean under prediction, both displayed in minutes. The scores are averages from five folds of testing data.

this case, we used the physics-based model described in Section 4.3.2 by itself to predict a subject's core temperature over time. These predictions were used in turn to predict expected pull-time, as described in Section 4.3. The results of evaluating the model against our objective functions from Section 4.3.6 are shown in table 4.2. This table also shows the results of other human-derived primitives that were applied to the feature space and objective functions. As described in Section 3.3, these machine learning functions are used by EMADE as primitives. By itself, the physics-based model dominates the kNN, decision trees, and Bayes classifier, while it is co-dominant with the OMP and SVM functions. When leveraged together, we were able to produce algorithms that dominate all of these single-element solutions.

A graph of the Pareto fronts over time of two objectives from our five objective optimization is shown in Figure 4.17. Because both objectives represent error metrics, they are both minimized. The inner-most (bottom left) line represents the final Pareto front computed in these two dimensions. Analysis of these scores illustrates the power of EMADE, which evolves starting with human-derived point solutions to reach Pareto-dominant solutions to human-generated algorithms. With a mean over prediction of 11.3 minutes and a

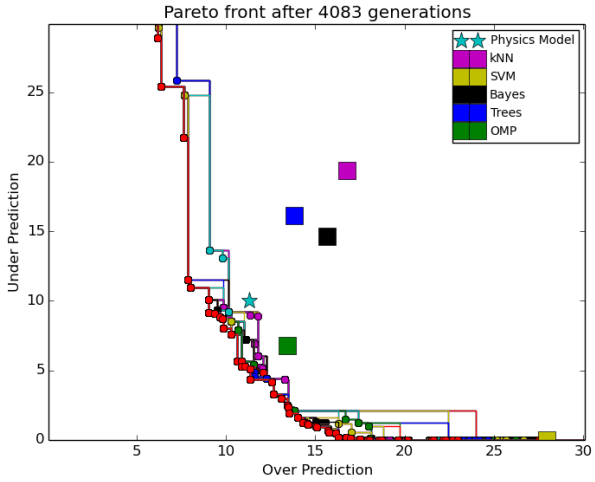


Figure 4.18: Pareto fronts of solutions generated by EMADE when projected from nine dimensions to two dimensions: mean over prediction and mean under prediction, both displayed in minutes. The scores are averages from five folds of testing data.

mean under prediction of 10.0 minutes for the physics-based model, there are numerous solutions on the final Pareto front to choose from. Each of the human-derived machine learners that are used as primitives in EMADE are dominated by our Pareto front in the plotted objectives.

Optimization 2

A second optimization was performed using the Pareto front of the previous optimization as a set of seeds. The Pareto front over time for this cycle is shown in Figure 4.18. A selection of solutions dominant to the physics based model is shown in Table 4.3.[5] EMADE's dominance of the physics-based model and the basic machine learners holds in the two objectives of interest.

Table 4.3 shows the ten solutions in the nine objective space generated by EMADE that dominate the physics-based model on all objectives. There are also results not shown in this table that dominate the human-derived machine learners on all objectives. The algorithm that was chosen for implementation corresponds to ID 1359 in the table. This algorithm's

[5]Depth, count over, and count under do not factor into dominance. We seek to minimize these objectives, but they do not have negative effects on performance.

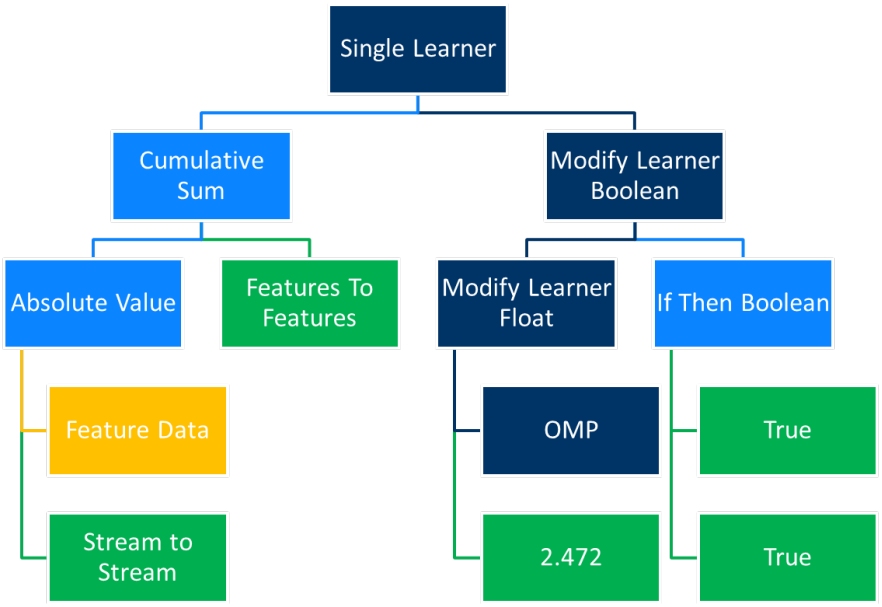


Figure 4.19: An algorithm that dominates the physics-based model. This result was chosen to be used in the field.

Table 4.2: 9 Objectives Baseline Performance. Errors are in minutes.

Method	Mean		Maximum		Standard Deviation	
	Over	Under	Over	Under	Over	Under
Physics Model	11.3	10.0	38.8	28.5	9.30	6.21
kNN	16.8	19.4	33.3	43.0	10.53	11.43
SVM	27.9	0.0	51.5	0.0	12.22	0.00
Bayes	15.7	14.6	38.1	24.1	11.09	7.37
Trees	13.8	16.1	40.7	28.9	11.62	9.65
Random Forest	16.9	13.3	42.9	29.8	12.15	9.47
k-means	0.0	42.8	0.0	60.0	0.00	16.74
BLUP	1,099.0	764.7	5,885.3	2,348.5	1,575.09	722.45
OMP	13.4	6.8	29.1	22.6	8.99	5.48

Table 4.3: 9 Objectives Machine Learner Performance. Errors are in minutes.

ID	Mean		Maximum		Standard Deviation		Depth of
	Over	Under	Over	Under	Over	Under	Tree
430	10.4	9.5	25.2	26.0	8.83	6.15	3
431	10.4	9.5	25.2	26.0	8.83	6.15	2
541	11.0	9.4	26.2	26.8	8.46	6.17	6
544	11.3	9.4	26.8	26.7	8.84	6.16	3
545	11.3	9.4	26.8	26.7	8.84	6.16	4
1359	10.9	8.0	28.1	22.9	8.68	6.01	3
1360	10.9	8.0	28.1	22.9	8.68	6.01	3
1361	10.9	8.0	28.0	22.9	8.67	6.00	5
1451	11.3	9.0	28.3	24.2	8.21	6.15	4
1834	10.3	7.6	31.6	22.3	9.24	6.18	4

tree is presented in Figure 4.19. We selected this algorithm over other options because it offers similar performance in over prediction error to the physics-based model, while significantly reducing under prediction error and maximum error in both the under and over predicted cases.

This section showed how EMADE elevated the capabilities of standard genetic programming techniques to produce solutions that outperformed human-generated algorithms. We leveraged vector-based feature construction and machine learning functions as primitives to improve the predictions made by a physics-based model, resulting in a safer algorithm for determining how long a first responder can spend in a hazmat suit. This algorithm will be distributed on mobile devices and deployed for use in pairing responders for tasks in hazardous environments.

4.4 Time Series Regression: LIDAR

So far we have shown EMADE on a categorical classification problem (Section 4.1), a time-series classification problem (Section 4.2), and a categorical regression problem (Section 4.3). The final application to complete this set is a time-series regression problem. This section uses EMADE to design a bathymetric LIDAR ranging algorithm.

Bathymetric LIDAR presents a good application for EMADE for several reasons. First, performance is extremely important. The tolerance on the entire LIDAR system must be

on the order of decimeters, obeying the rule-of-thumb

$$\text{Tolerence}(d) = \sqrt{0.5^2 + (0.013d)^2} \text{ meters}, \tag{4.4}$$

where d is the depth in meters [55]. The error per sample of the ranging algorithm is

$$\text{Error}_{\text{OPL}} = c_{\text{water}} \cdot \frac{1}{F_s} = \frac{225,000,000 \text{ meters}}{1 \text{ second}} \cdot \frac{1 \text{ second}}{1.6 \cdot 10^9 \text{ samples}} = .14 \frac{\text{meters}}{\text{sample}}, \tag{4.5}$$

where $\text{Error}_{\text{OPL}}$ is the error in optical path length. To convert this into a depth error we must factor in the angle the beam enters the water,

$$\text{Error}_{\text{Depth}} = \cos 20^\circ \cdot \text{Error}_{\text{OPL}} = .94 \cdot .14 \frac{\text{meters}}{\text{sample}} = 0.1315 \frac{\text{meters}}{\text{sample}}. \tag{4.6}$$

This error rate means that at our shallowest depths (i.e. $d \approx 0$ m), we may only make an error of three range bins before going over the error budget of our entire system.

Second, the multiple objective optimization is extremely useful for bathymetry applications. Understanding the trade-offs between under-prediction and the more dangerous over-prediction (which can cause a ship to run aground) is essential. Another useful objective is the probability of miss, a non-linear function where a miss is categorized by any error outside the tolerance defined in Equation 4.4.

Finally, this application has a well-documented algorithm to use to seed the evolutionary process known as the Interest Point Detection algorithm [56]. This algorithm consists of three steps. The first is to condition the waveform using a Savitzky-Golay polynomial filter. Second, we extract the peaks from the waveform. Finally, we use the peaks to conduct an informed search, where we process small windows preceding the peaks for inflection points by studying the zero crossings of the second derivative. We compute the OPL by finding the difference in samples between the two inflection points representing the sea surface reflection and seafloor reflection. The inflection points are used instead of the peak locations

Table 4.4: Parameter distribution for lidar waveform simulation

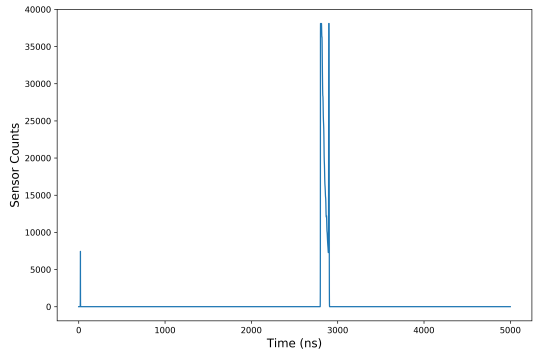
Parameter	Distribution	Units
Average laser power	$\mathcal{N}(\mu = 30, \sigma = 0.9)$	W
Full-width half-max	$\mathcal{N}(\mu = 1.7, \sigma = 0.133)$	ns
Off nadir angle	$\mathcal{N}(\mu = 20, \sigma = 0.0673)$	deg
Filter spectral width	$\mathcal{N}(\mu = 1.4 \times 10^{-9}, \sigma = 0.004\,67)$	nm
Scan angle	$0.00197 * \mathcal{U}(\text{lower} = 0, \text{upper} = 183 \times 10^3)$	deg
PMT bias voltage	$\mathcal{N}(\mu = 550, \sigma = 9.167)$	V
Lowpass filter frequency	$\mathcal{N}(\mu = 6.14 \times 10^8, \sigma = 2.047)$	MHz
Latitude	$\mathcal{U}(\text{lower} = 4, \text{upper} = 25)$	deg
Longitude	$\mathcal{U}(\text{lower} = 104, \text{upper} = 124)$	deg
Water depth	$\mathcal{U}(\text{lower} = 0.25, \text{upper} = 55)$	m
Height above sea level	$\mathcal{N}(\mu = 400, \sigma = 10)$	m
Seafloor reflectance	$\mathcal{U}(\text{lower} = 0.01, \text{upper} = .25)$	
Seafloor tilt	$\mathcal{U}(\text{lower} = -20, \text{upper} = 14)$	deg
Wind speed	$\mathcal{U}(\text{lower} = 0, \text{upper} = 10)$	$\frac{m}{s}$
K_D	LogUniform$(\text{lower} = 0.06, \text{upper} = 10)$	$\frac{1}{m}$
β_π	$\mathcal{U}(\text{lower} = 0.001, \text{upper} = 0.003)$	$\frac{1}{\text{m·sr}}$
σ_{β_π}	$\mathcal{U}(\text{lower} = 8 \times 10^{-5}, \text{upper} = 4 \times 10^{-4})$	$\frac{1}{\text{m·sr}}$

because the peaks are more susceptible to stretching, saturation, and noise effects than the location of the inflection points. We translate the OPL directly to depth by multiplying by the angle the beam enters the water as we did for error in Equation 4.6.

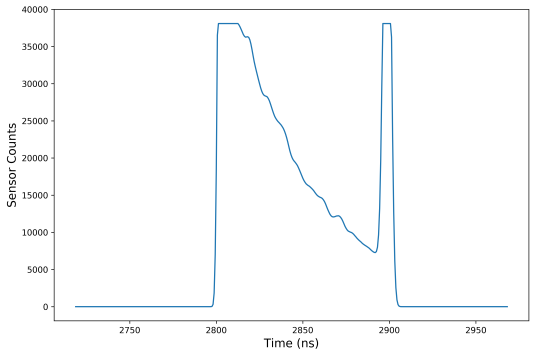
Before running the evolutionary process, we created a robust simulated bathymetric LIDAR dataset using a simulator [57]. The dataset comprised various depths, turbidity, noise levels, and wave speeds. A full cross section of parameters is shown in Table 4.4. The dataset was meant to reflect the best estimate of the real-world environment. In this case, the system parameters reflected a particular LIDAR configuration, and the environmental parameters reflected the characteristics of the South China Sea. Figure 4.20 displays an example LIDAR return produced by the simulator.

Next, to ensure we were able to seed the optimization with the current algorithm, we broke down the interest point method into the following primitives:

- Cut data lead: This method removes a fixed amount of leading data, which is con-



(a) Full bathymetric LIDAR waveform, including pulse as beam exits the system.



(b) Zoomed to enhance region with sea surface and seafloor peaks.

Figure 4.20: Example waveform produced by the bathymetric LIDAR simulator.

trolled by a parameter. This removes the large amount of the LIDAR return that precedes the sea surface reflection.

- Rebase: Rebase removes the mean from the signal.
- Savitzy-Golay filter: This filter fits a moving window of data to a k-th order polynomial. The window size and order of the filter are inputs.
- Peak detector: The peak detector looks for maxes (or mins) in the signal based off of an input delta. This parameter controls how low (or high) the signal must move from its peak before it is locked in as a local max. This reduces small peaks that occur from noise.
- Informed search: The informed search consumes a signal, a set of locations to perform a search from, and a window to look at preceding each location. The informed search computes the second derivative of each window and returns the zero crossing locations.
- Depth estimate: This method consumes a set of locations and a sampling rate and returns the distance in meters between the first and the last locations.

Each primitive was implemented with a set of input parameters that could be tuned to

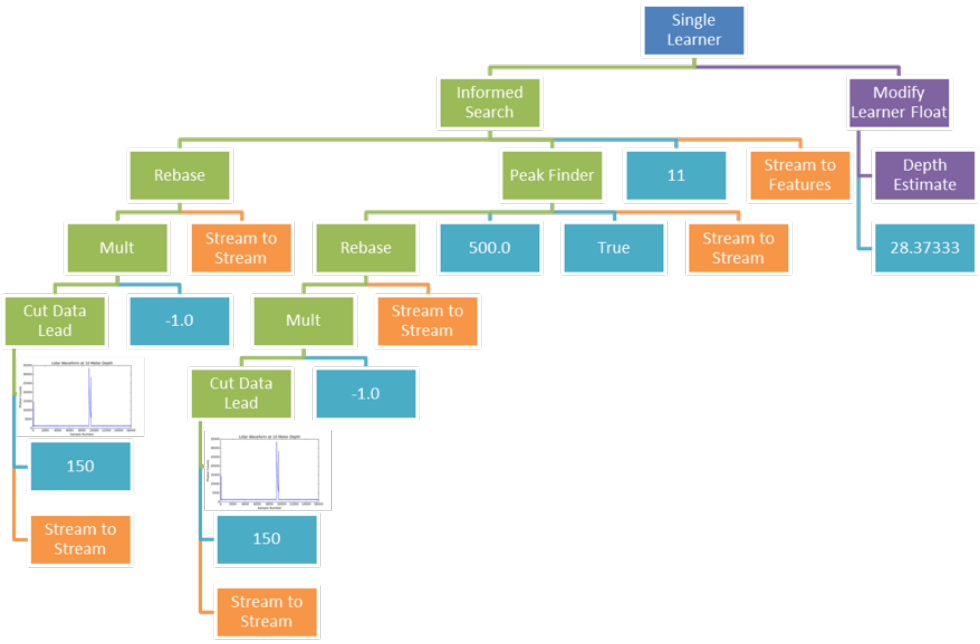


Figure 4.21: The interest point method for LIDAR signal processing implemented in an EMADE tree structure to seed the optimization.

change the performance. The process of decomposing the seed and implementing its components in EMADE enables the framework to have a set of problem-specific tools it can leverage.

Figure 4.21 shows an example tree structure implementation of the interest point method without the Savitzky-Golay filter included. This and several variants were used to seed the optimization.

4.4.1 Optimization Strategy

Figure 4.22 shows how the training dataset was broken up by several factors. First, the 10,000 instances were filtered down by only selecting those where the product $K_D \cdot Depth < 4$. This is a well-established value in the community for conditions where a bottom return should be visible. Of the 10,000 waveforms that comprised the original dataset, 2,298 remained in the training set. Next, the data was split yet again based on whether or not the interest point method was able to successfully detect at least two peaks in the waveform, since if we include cases where the interest point currently fails, error quantification

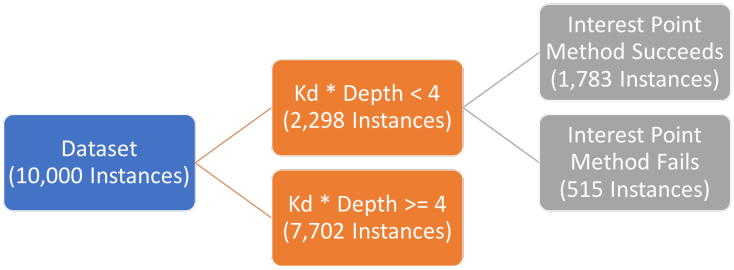


Figure 4.22: Dataset partitioning for optimization.

becomes a challenging and somewhat arbitrary process. Of the 2,298 waveforms where a bottom return should be detectable, the interest point method found a bottom in 1,783, leaving 515 without predictions. From this data, we construct three different algorithm types, to be used in a processing structure such as Code Listing 4.1.

Code Listing 4.1: Pseudo-code for algorithm processing pipeline.

```
if interest point method predicts bottom then:
    return prediction using algorithm type 1
elif algorithm type 2 predicts K_D*Depth<4:
    return prediction using algorithm type 3
else:
    return "No bottom detected"
```

4.4.2 Primitives

We implemented the following methods from the literature as primitive functions in EMADE:

- Richardson-Lucy Deconvolution [58, 59]
- Gaussian Decomposition [60, 61, 62]
- Matched Filtering
- B-Spline [63]
- Supersampling

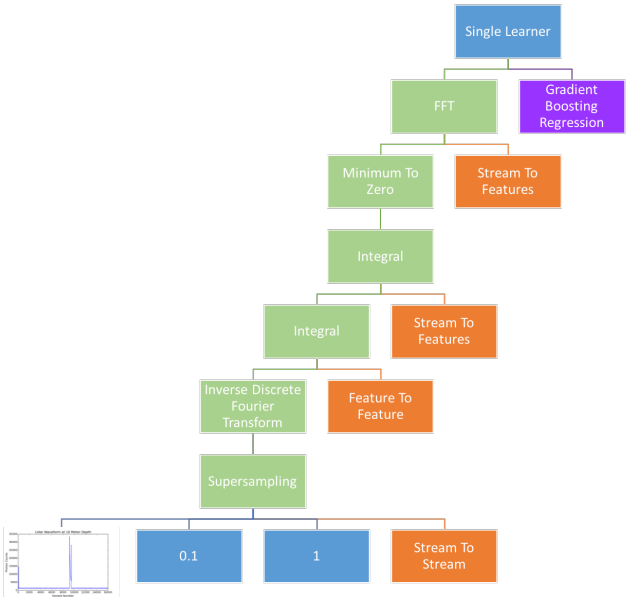


Figure 4.23: A seed algorithm that combines a processing algorithm from literature and machine learning.

- Gaussian Fitting [64, 65]
- Lognormal Fitting [64, 65]
- Quadrilateral Fitting [66, 67]

4.4.3 Optimization 1 – Refining Interest Point Algorithm

Our first optimization was on the data where the interest point method currently succeeds. We cross-folded the 1,783 instances in the dataset into five 80/20 train/test splits. EMADE uses these splits to protect against overtraining. EMADE scores each algorithm based on its average across all five splits. For each evaluation of an algorithm, the machine learning models are fit on the training portion, while being scored against the testing portion.

We seeded the first optimization with the interest point algorithm along with a hand-constructed algorithm that combines one of the LIDAR primitives (Supersampling) along with a spectral representation and a gradient boosting regression machine learner. Figure 4.23 shows this hybrid seed algorithm.

EMADE ran on a two-tiered dataset. The first tier was a small subset of one cross-folded pair (100 training samples and 50 test samples) that was used as a litmus test to determine if the algorithm would be successful. While an evaluation on the entire five folds takes an average of 1359.93 seconds per individual, the smaller dataset takes only an average of 64.5 seconds per individual. EMADE evaluated 11,902 individuals on the smaller dataset without needing to promote them to the second tier, while choosing to fully evaluate 2,455 algorithms on all five folds. In processing time, EMADE saved over four thousand CPU-hours of computation at the expense of 43.99 CPU-hours of litmus testing the successful individuals. EMADE minimized 8 objectives for this optimization:

- RMS Error
- Probability of Miss
- Percent Error
- Over Prediction Error
- Under Prediction Error
- False Negative Bottom
- Number of Invalid Individuals
- Number of Elements in Tree

EMADE used an initial population size and elite pool size of 514 individuals. The initial population contained 2 seeded and 512 randomly initialized algorithms. Each generation, EMADE produced 300 offspring. Table 4.5 shows the mating and mutation methods used, along with the probabilities of application.

We harvested results after EMADE ran for 57 generations and in that time evaluated over 14,000 individuals. Figure 4.24 shows the non-dominated frontier on two objectives. Across all 8 objectives, 73 individuals were non-dominated solutions. One of EMADE's

Table 4.5: EMADE Mating and Mutation Rates.

Operation	Operation Type	Probability
Crossover	Mating	0.5
Crossover Ephemeral	Mating	0.5
Headless Chicken Crossover	Mating	0.1
Headless Chicken Ephemeral Crossover	Mating	0.1
Insertion	Mutation	0.05
Insert Modify	Mutation	0.05
Ephemeral	Mutation	0.25
Uniform	Mutation	0.05
Shrink	Mutation	0.05

strengths is its ability to search this high-dimensional trade-off space to create algorithms that can be used under a number of different circumstances depending on an a posterior decision vector of objective importance. Figure 4.25 and Figure 4.26 show the gold and cyan algorithms from Figure 4.24, respectively.

We further analyzed the algorithm in Figure 4.25 by evaluating the variance of optical path length estimation against water depth for a K_D of 0.14 1/m. Figure 4.27 shows the comparison of the evolved solution and the interest point method, with the EMADE solution strongly outperforming the state of the art.

Both evolved solutions in Figure 4.25 and Figure 4.26 behave similarly. They evolved around primitives in the EMADE toolbox from computer vision, erosion ellipse and erosion cross. Both of these methods sharpen images, but here they instead are able to convert a LIDAR return into a rectangle function because they treat the signal as a $1 \times N$ image. The rectangle starts at the surface and ends at the seafloor. The orthogonal matching pursuit in each algorithm correlates the area of these boxes to a depth. Because the boxes have fixed heights from these methods, the area is directly proportional to the depth.

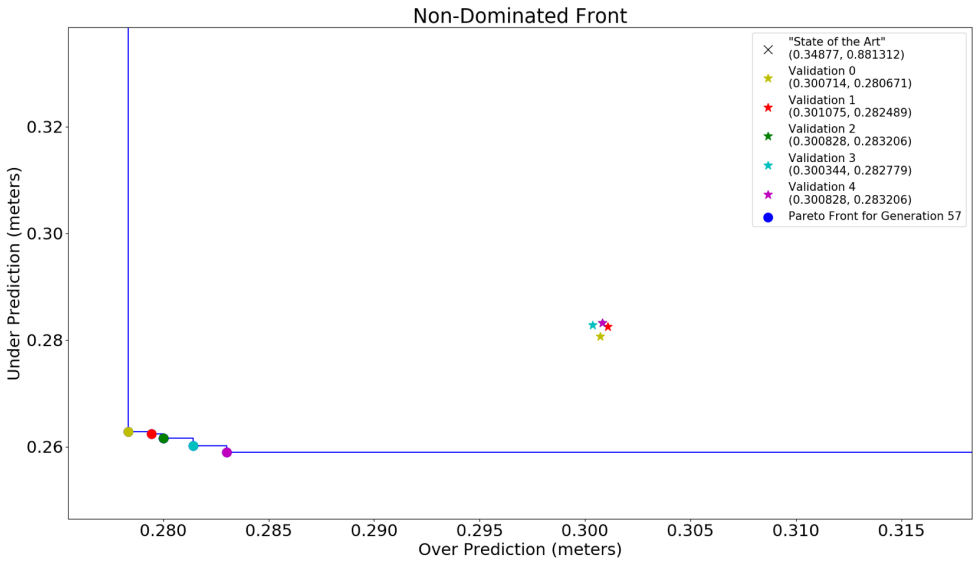


Figure 4.24: Non-Dominated frontier after 57 generations with respect to under prediction and over prediction errors. The stars show the performance of each non-dominated algorithm on the held-out validation set.

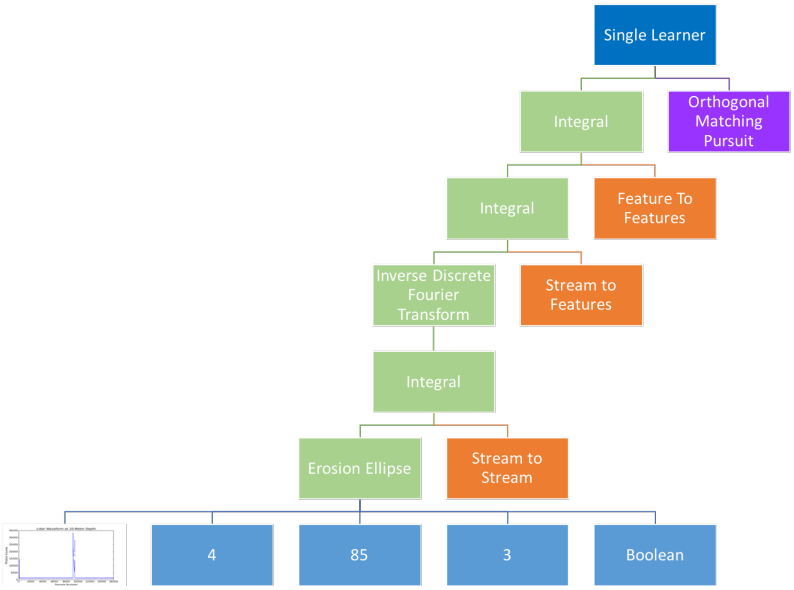


Figure 4.25: An EMADE evolved solution with validated over prediction error of 0.30 meters and validated under prediction error of 0.28 meters.

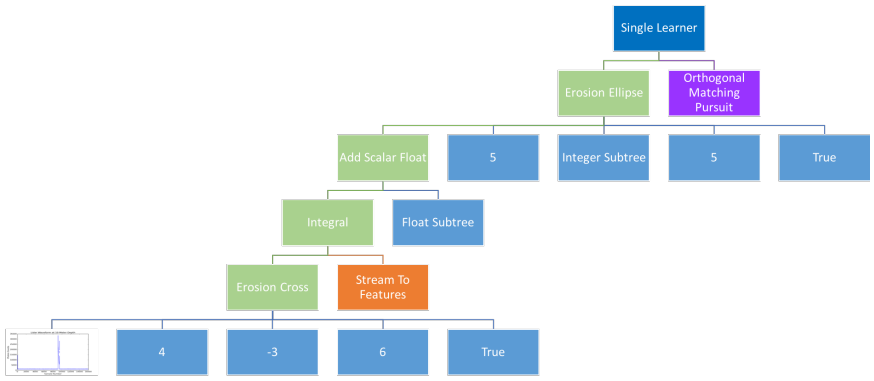


Figure 4.26: An EMADE evolved solution with validated over prediction error of 0.30 meters and validated under prediction error of 0.28 meters.

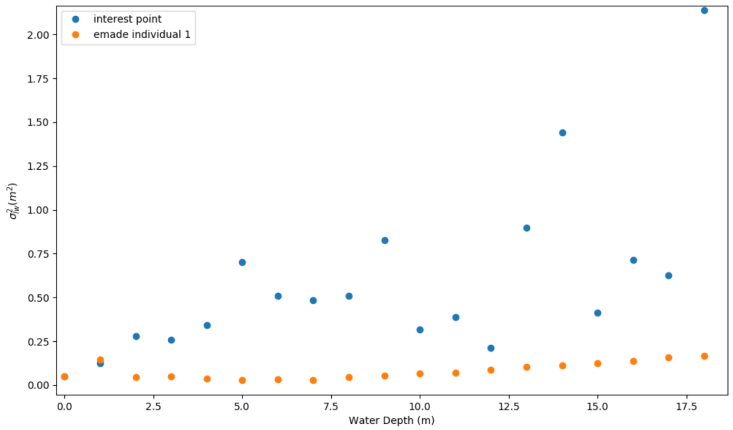


Figure 4.27: Variance vs Depth for evolved solution and interest point method.

4.4.4 Optimization 2 – Classification of Detectability

If the interest point method fails to make a prediction, it is usually for one of two reasons. Either it fails because the peak from the seafloor is indistinguishable from the peak of the sea surface due to a shallow seafloor, or it fails beacuse the peak from the seafloor is too small to be distinguished from noise due to the seafloor depth. In both of these cases, we would like to know, prior to processing, if we should be able to see a bottom due to the murkiness of the water. Our second optimization focused on classifying if waveforms should be visible or not based on the $K_D \cdot Depth < 4$ metric.

For this problem, we constructed a dataset by randomly sampling 500 instances where $K_D \cdot Depth < 4$, along with 500 instances where $K_D \cdot Depth \geq 4$. We constructed our dataset this way so that our classes of 0 (≥ 4) and 1 (< 4) would be balanced. We again used a two-tiered structure, the first tier being 200 waveforms, the second being 800. We split the 200 waveforms into 150 for training and 50 for testing. We cross-folded the 800 waveforms into five 80%/20% partitions.

This optimization used three objectives: false positive rate, false negative rate, and the number of the elements in the genome. A false positive in this context is a classification that a waveform should be predictable but is not. A false negative is a classification that a waveform should not be predictable when in fact it is. Figure 4.28 shows a classification algorithm from the optimization. This algorithm had a false positive rate of 10% and a false negative rate of 13.8%. The overall accuracy of the classifier is 88.1%.

4.4.5 Optimization 3 – Predictions Without Peaks

The final optimization we ran was the case where the interest point algorithm failed to detect at least two peaks in the waveform, thus making depth prediction impossible. For the third optimization, we cross-folded the 515 instances of the 10,000 where $K_D \cdot Depth < 4$ and the interest point failed, using five folds with an 80%/20% split.

Figure 4.29 shows our non-dominated frontier after 60 generations. We do not show

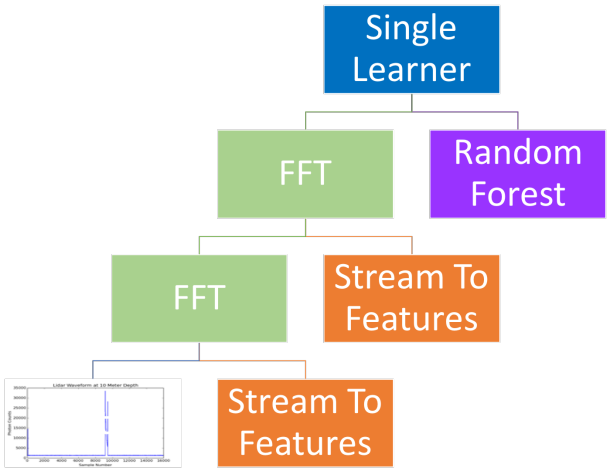


Figure 4.28: Binary classifier for predicting if $K_D \cdot Depth < 4$.

performance of the state-of-the-art interest point algorithm here, because this is the data on which it failed to predict and thus could not be scored. Figure 4.30 shows the algorithm represented in cyan in Figure 10. Similar to the algorithm evolved in Optimization 1, this algorithm uses an erosion function, but also combines this with a spectral representation and a support vector machine to regress the depth. This algorithm averaged 0.359 meters of over-prediction and 0.357 meters of under-prediction errors. While the performance of these algorithms is lower than those from Optimization 1, these are inherently more challenging waveforms.

4.4.6 Conclusions

This problem represents the first time-domain regression problem solved using a genetic programming based autoML framework. The evolved solution is a simpler and more elegant solution than what has been used in industry for decades. The three algorithms developed in this section afford a significant performance improvement over the interest point method. Where the interest point method succeeded before, the evolved solution validated to 0.30 meters of average over-prediction and 0.281 meters of under-prediction. On the same dataset, the interest point method obtained an average over-prediction of 0.348 me-

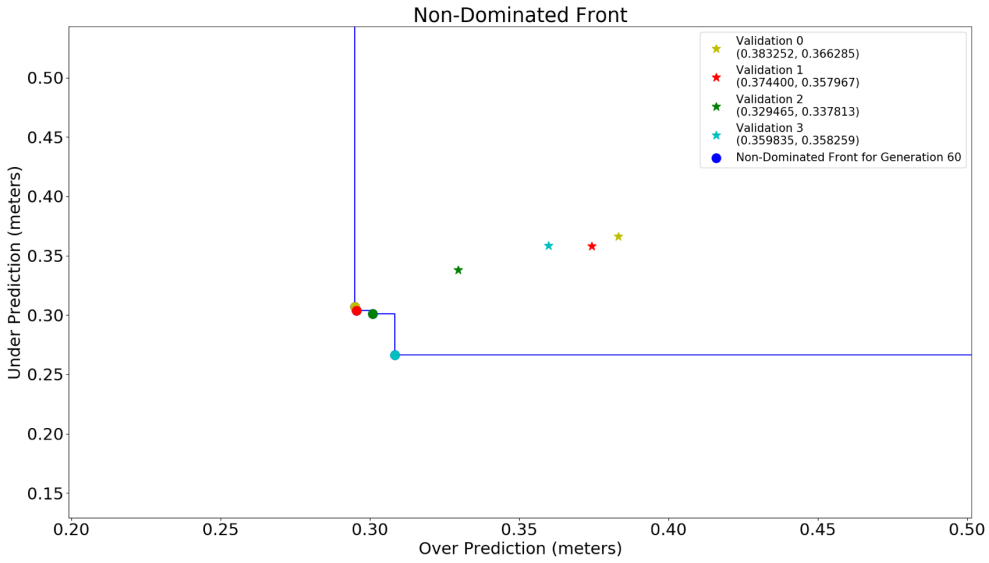


Figure 4.29: Non-dominated frontier for predicting depth where interest point method fails.

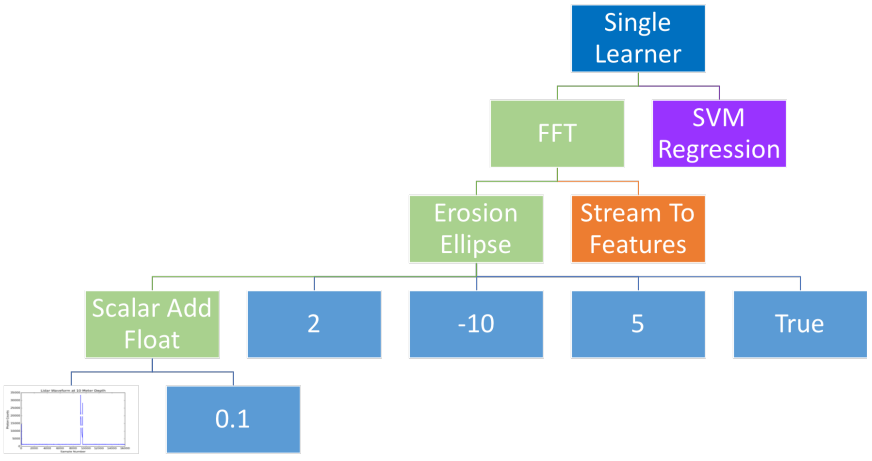


Figure 4.30: Evolved algorithm for predicting depth when interest point method fails, but the binary classifier from Optimization 2 predicts $K_D \cdot Depth < 4$.

ters and an average under-prediction of 0.881 meters. Thus, the evolved algorithm achieved a 13.8% reduction in over-prediction and a 68.1% reduction in under-prediction errors. The variance versus depth is also significantly better.

In addition to improving performance where the current algorithm can detect a seafloor return, we also evolved a solution that enables ranging where it was previously impossible, with a validated performance of 0.38 meters over-prediction and 0.366 meters under-prediction. The evolved solution's performance where the interest point method fails, compared to the successes of the interest point method, is only 9.2% worse in over-prediction, and 58.5% better in under-prediction.

Finally, through the evolution of the binary classifier in conditions where the interest point method fails, we achieve an 88.1% accuracy in our ability to predict with this evolved solution. Prior to this evolved classifier, the interest point method was failing to predict on 515 out of 2,298 instances, representing an accuracy of 77.6%. Thus, the evolved classifier picks up 46% of possible improvement, i.e.

$$\frac{\text{new} - \text{previous}}{1 - \text{previous}}.$$

Overall, this section shows the power of autoML coupled with high-fidelity simulation to outperform human-developed state-of-the-art. Our most substantial performance gains came from non-ideal signals that the interest point method encountered, such as saturation causing peaks to disappear (and thus making the detection of interest points fail), or spurious noise peaks causing false seafloor returns, resulting in significant under-prediction. The evolution found that the best way around these problems was to look at the overall energy level of the signal until it fell below a certain threshold, ignoring the spurious or saturated peaks that may fall between the sea surface and seafloor.

4.5 Feature Classification: Titanic Dataset

We placed EMADE in direct competition with human researchers, leveraging the Titanic dataset found on Kaggle.com, which Kaggle uses for its "introduction to data science" tutorials. We gave 39 Georgia Tech students, enrolled in the Vertically Integrated Project (VIP) class for Automated Algorithm Design, ranging from sophomores to seniors, one week to develop algorithms to solve this problem. The students were grouped into small teams and asked to submit co-dominant solutions, as a method to reinforce the concept of multiple objectives. Figure 4.36 displays the results of the students' efforts evaluated on the validation data from Kaggle.

Similar to the adult dataset used in Section 4.1, the Titanic dataset is also feature data. There are nine features used to predict a binary outcome: whether a passenger died or survived the sinking of the Titanic. To prepare the data for EMADE, we preprocess the same training data used by the students and cross-fold the set ten times using a 90%/10% split for training and testing within EMADE. We use three optimization objectives: false positives, false negatives, and the number of elements in the tree structure. To create the truth data for this problem, we define a positive test case to be survival of the disaster. In the unsplit training data, there are 342 positive test cases (i.e. survivors) and 549 negative test cases (i.e. non-survivors).

We configure EMADE to run with a population size of 512 individuals and initialize it with random genomes. We use four types of crossover and apply them independently, with the rates of 50% for single-point, 50% for ephemeral, 10% headless chicken, and 10% headless chicken ephemeral. For mutation, we use six methods, and also apply them each independently. We use the rates of 5% insertion, 10% insert modify, 25% ephemeral, 5% node replace, 5% uniform, and 5% shrink. For selection, we use NSGA-II on the combined population of current offspring and non-dominated individuals.

We ran EMADE on a cluster of 14 Linux machines for an eight-day period, evaluating

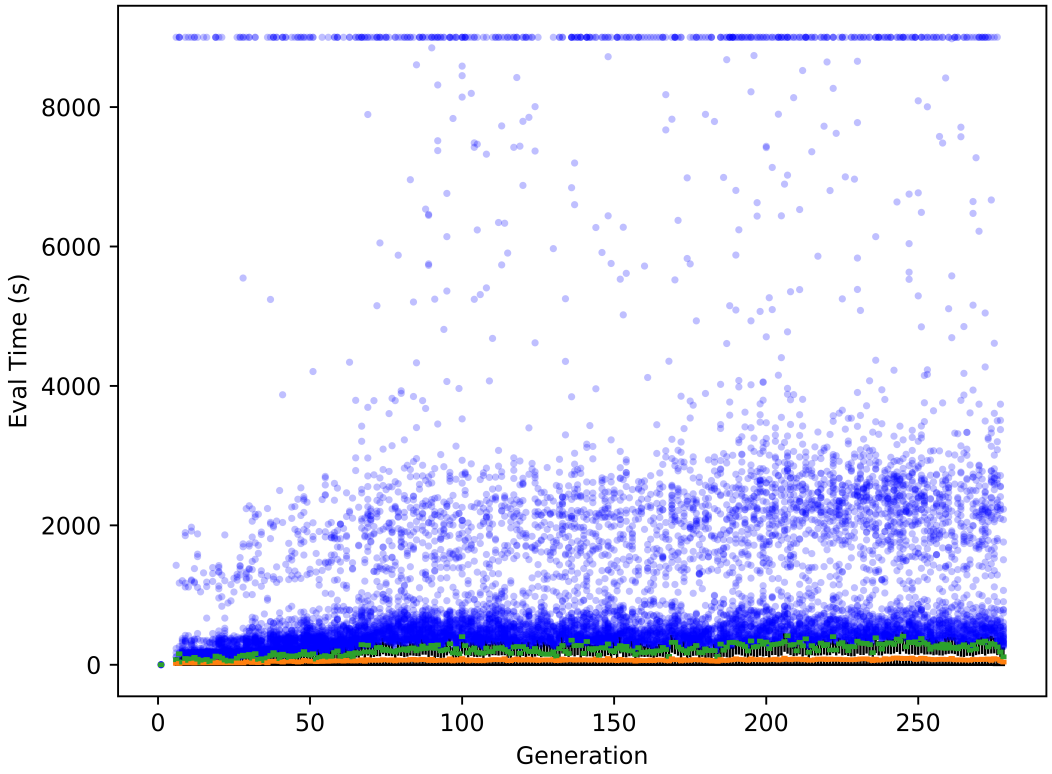


Figure 4.31: Average evaluation time across all ten cross-folds of training data per generation.

88,785 individuals in total. EMADE completed 277 generations and found 1,915 unique non-dominated individuals over the course of the optimization. At the end of generation 277, 336 individuals remained on the non-dominated frontier for all three objectives. The total compute time, when aggregated across all workers, was approximately 5,295 compute hours. The average evaluation across all ten folds of data lasted 214 seconds. Figure 4.31 presents a box plot of the evaluation times per generation of the same nature as in Section 4.1. We see from this plot that the average evaluation time increases as EMADE finds a higher rate of successful individuals and more complex solutions with time.

Figure 4.32 shows the time between each generation during this optimization. This problem deals with a relatively small amount of data, only 891 instances. For more typical problems, we would expect these times to be much larger. Even on this small dataset, however, we can see that there is plenty of time in which we can run our evolutionary loop

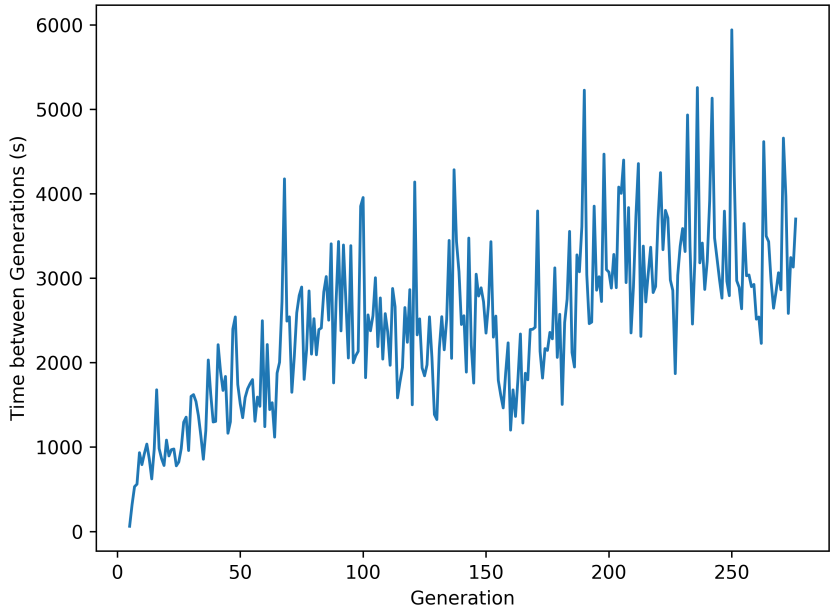


Figure 4.32: Inter-generational time throughout the optimization.

to prepare the next generation. In general, we should take advantage of as much of this time as possible to have a better probability producing successful offspring. Chapter 5 explores novel methods to increase this probability.

We can analyze the results of EMADE and human experts with the area under the curve (AUC) of the non-dominated frontier computed from false positives and false negatives using Equation 4.1. Figure 4.33 plots the AUC computed from test data during the optimization process. Figure 4.34 shows The AUC % reduction calculated using the equation

$$\% \text{ Reduction} = 100 \cdot \frac{\text{previous} - \text{new}}{\text{previous}}, \tag{4.7}$$

for each generation. For both graphs, there is a large amount of activity in the initial 60 generations before it begins to taper off.

Figure 4.35 presents the number of non-dominated individuals at each generation. Given that the population size for this optimization is 512, we see that our non-dominated frontier never exceeded the size of our population. It did, however, continue to grow throughout the optimization process. Considering that the AUC (computed on false positives and false

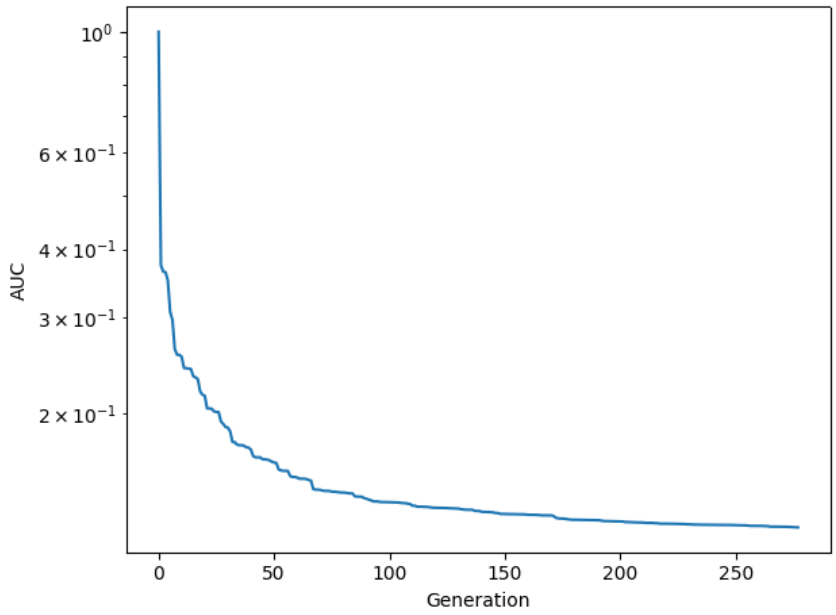


Figure 4.33: Area under the curve for each generation of EMADE plotted on a log scale.

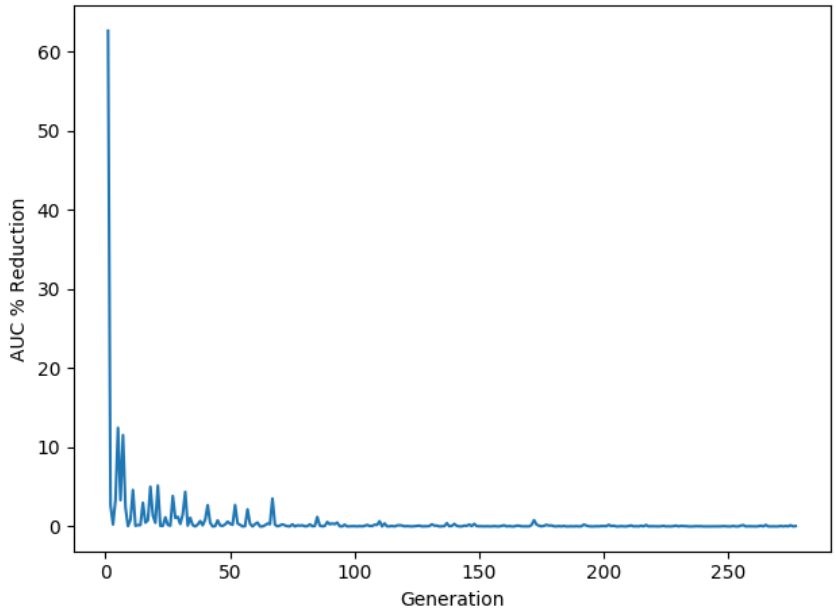


Figure 4.34: Percent reduction in area under the curve from one generation to the next.

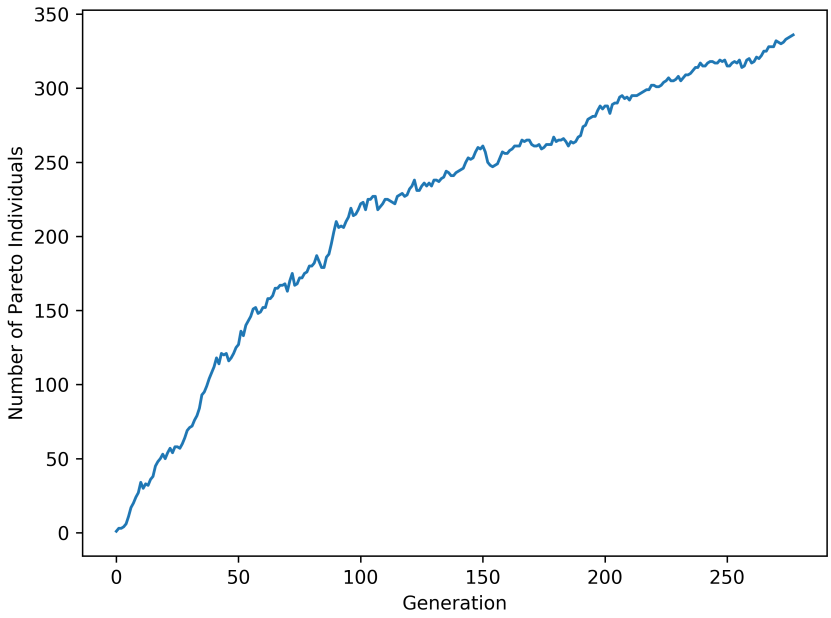


Figure 4.35: Number of non-dominated individuals present at each generation of the optimization

negatives) was not decreasing at a high rate, a large contribution to the number of non-dominated individuals is the number of elements present in the tree. This growth of the non-dominated front informs us that we maintained some genetic diversity with multiple methods of varying complexity computing similar objective scores.

After the evolutionary process, we evaluated 336 non-dominated individuals on the held-out validation data. The validation data consists of 158 positive test cases (survivors) and 260 negative test cases (non-survivors). Figure 4.36 shows the individuals that remained Pareto-optimal on the objectives of false positives and false negatives and the resulting non-dominated frontier.

The result of our optimization with EMADE was a non-dominated frontier that completely dominated all student-produced algorithms except for one. Ironically, upon further investigation, the one remaining student had preprocessed his data in an unintentional way. In both the training and the testing sets, there was one observation each with a particular feature that was missing a value, he tried to drop both records from his algorithm, but in-

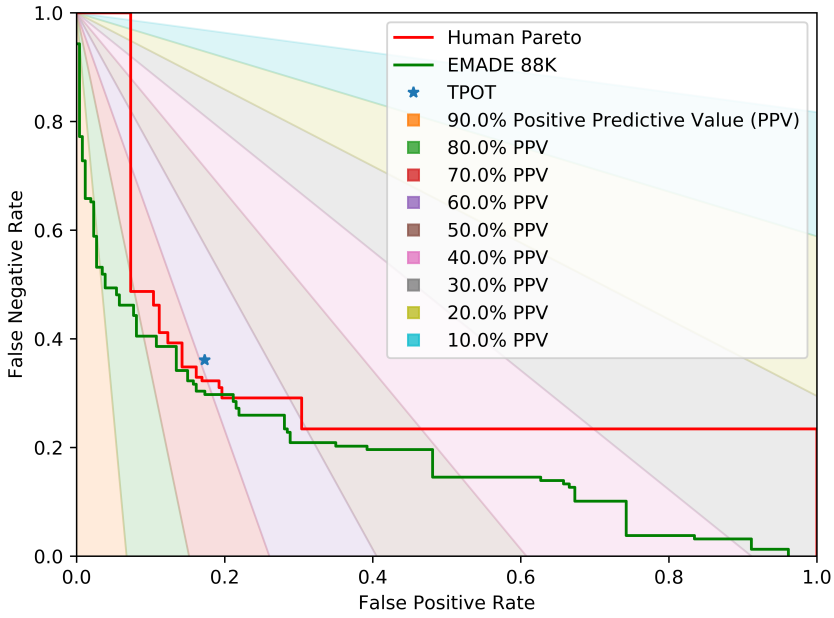


Figure 4.36: Comparison between non-dominated frontiers found by Gerogia Tech students and EMADE.

stead accidentally imputed a feature for the missing value. These two records happened to be survivors, and the synthetic feature ended up linking these two samples together, resulting in a unique point that no other algorithm was discerning without picking up additional false positives. While one could attribute this to luck, it does speak to the necessity of more feature imputation methods within EMADE, as this student's solution did not exist in EMADE's search space.

On the validation data set, EMADE achieved an AUC of 0.19, while the AUC from the team of human experts was 0.32. This difference in AUC represents a reduction of 40.63% in AUC as computed in Equation 4.7. The most substantial improvements came along the boundaries of the Pareto frontier, as shown in Figure 4.36.

As a comparison, we ran TPOT using ten cross-folds, and its default evolutionary settings: a population of size 100, a probability of crossover of 10%, and a probability of mutation of 90% for 100 generations. TPOT uses multi-objective optimization, like EMADE; however, unlike EMADE, it is limited to two objectives and one is always the complexity

of the individual. This limited objective space means that the trade-off space is not fully explored between types of errors. At the end of 100 generations, TPOT returned seven non-dominated individuals. TPOT returns an export of the best individual by accuracy (choosing to ignore the objective of complexity). We validated the performance of the optimized pipeline for accuracy on the same validation set as EMADE and marked the result with a star on Figure 4.36. The classifier produced by TPOT had an overall average accuracy of 84.18% on the cross-folded data, but 75.60% on the held out validation data, pointing toward some overtraining. Even though EMADE evaluated almost nine times more individuals than TPOT, the non-dominated frontier EMADE produced was less susceptible to overtraining, because of the robustness of multiple objectives.

There is a danger of overtraining within EMADE. In some cases, we observed better performance on validation data from earlier generations of EMADE than later generations. This loss of performance informs us that on the cross-folded training data, we have found individuals that have outperformed existing solutions in both objectives, knocking individuals that were performing better on the validation data out of the non-dominated frontier.

This danger of EMADE illuminates our data as well; there are statistical differences between our test set and our validation set. Problems like the Titanic data set are susceptible to these issues. Naturally, machine learning should fair better on problems where the features have a more correlated impact on the results, and because EMADE uses machine learning, it has the same issues. EMADE is particularly problematic as many algorithms have underlying random states that can virtually build a random number generator into EMADE; theoretically, given enough time it can generate enough random individuals that the exact correct sequence of predictions is generated without ever learning the features.

In conclusion, this demonstration shows EMADE's ability to traverse the search space of a feature classification problem efficiently, and outperform both human experts as well as other automated machine learning platforms. The three leading enabling technologies that allow EMADE to outperform are its use of multiple objectives to maintain a diverse

population, as well as its high-level primitives and free-flowing tree structure.

4.6 Two-Dimensional Classification: Imagery

A large portion of Intelligence, surveillance, and reconnaissance (ISR) is the identification of potential points, persons, or objects of interest in feeds of imagery. Screening this imagery is a mundane and repetitive task currently handled by scores of analysts hand-annotating image data. Machine learning can supplement the analysts by serving as a prescreening tool, drawing attention to objects of interest. Because much of their burden is relieved, an analyst can be more productive and less prone to errors.

Researchers from the fields of computer vision and deep learning have used image processing techniques to detect, recognize, and identify objects of interest within overhead imagery. However, as with all machine learning problems, researchers tend to have a limited scope of methods to apply (based on their background) and a limited amount of time to tune their models.

This section details an object detection approach for the xView Detection challenge (http://www.xviewchallenge.org), a machine learning competition by Defense Innovation Unit Experimental (DIUx) and the National Geospatial-Intelligence Agency (NGA). The dataset consists of 30cm satellite imagery with 60 object classes annotated with bounding boxes, totaling over 1M object instances. The competition centers around new algorithms to aid in national security and disaster response activities.

4.6.1 Adding Computer Vision Primitives

Section 3.5 showed a number of image processing techniques implemented from OpenCV; however, most of these are not state-of-the-art features for object detection or recognition within imagery. After a brief literature review, we implemented the following techniques as primitives in EMADE:

- Histogram of oriented gradients (HOG) [68]

- DAISY features [69]
- Grey level co-occurrence matrices (GLCM) [70]
- Batched stochastic gradient descent (SGD)
- Batched passive aggressive classifier
- Deep neural network classifier (DNN)

Capabilities like HOG, DAISY, and GLCM, are important for EMADE because they allow traditional machine learning methods (i.e. non-DNN) to operate effectively on image data. They can operate effectively because these techniques are able to find useful features within images that are robust to scaling, translations, and rotations.

4.6.2 Processing Data

This section represents the first foray into creating machine learning algorithms for imagery with EMADE. As such, we bounded the task to the creation of a binary classifier of image chips. From the xView dataset, we created 224x224 pixel image chips. We chose this size to support the implementation of pre-trained deep neural networks (that are configured to these dimensions) as primitives. For each chip, we labeled the chips that contained buildings as positive test cases and those that did not contain buildings as negative test cases.

4.6.3 The Competition

The current leading algorithms for image classification from large datasets are deep neural networks. To compare performance with DNNs, we trained a Resnet v2 101 model [71] on an equal amount of building and background images. Since this a traditional classification task, we did not have to make major modifications except for balancing the data, which is important for effective learning with a DNN. This Resnet architecture achieved an accuracy

Table 4.6: Resnet Confusion Matrix for Classifying Buildings

	Not Building	**Building**
Not Building	8455	648
Building	869	8228

of 91.6%, a true positive rate of 90.5%, and a precision of 92.6%. Table 4.6 shows the confusion matrix for the trained Resnet architecture.

4.6.4 EMADE Results

We ran EMADE on a subset of the xView dataset to produce a binary classifier. We constructed a two-tiered dataset structure to allow the individuals to fail fast. The first tier contained 300 training images and 100 testing images. The second tier contained 3200 training images and 800 testing images.

An evaluation on the first-tier dataset took EMADE on average 72.53 seconds, while the second-tier averaged 522.31 seconds. Out of 5,270 candidate algorithms, EMADE found 3053 that did not perform well enough to promote to the next tier. This means the tiered dataset structure saved 381 CPU-hours of processing time at the expense of 44.66 CPU-hours of redundant computation on successful individuals. Over the course of the optimization, EMADE ran for a total of approximately 430 CPU-hours. Without a tiered structure, EMADE would have taken 765 CPU-hours. Therefore, the tiered dataset structure offered a savings of about 44 percent.

For the xView problem, we also compiled statistics on the success of the caching implementation. Reuse of existing cached results from primitives saved 68 CPU-hours of processing time. This savings came at a cost of initially storing the cached results, which took 10 CPU-hours to process the 4000 images. Our caching implementation netted 58 hours, which represents about a 12% improvement overall in EMADE throughput.

Figure 4.37 shows the non-dominated frontier produced by EMADE after evaluating over five thousand candidate algorithms. Note the Resnet DNN architecture described in

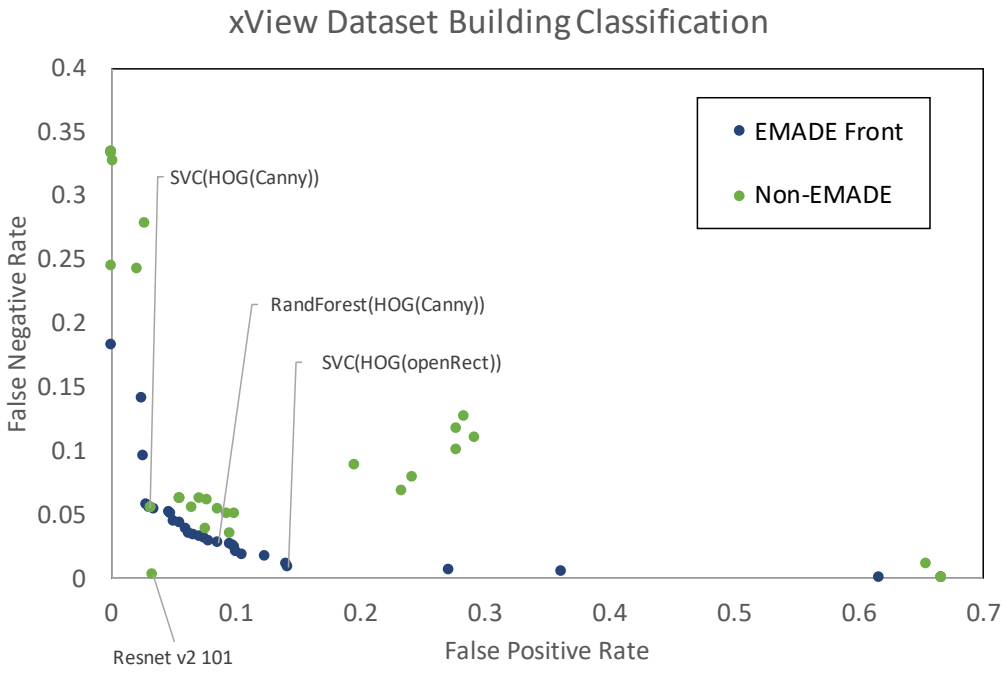


Figure 4.37: Non-dominated frontier of xView binary classification for buildings in EMADE.

Section 4.6.3 received a score of 0.0375 false positive rate and 0.0025 false negative rate. This DNN outperforms a great deal of the more traditional computer vision techniques in EMADE, and was co-dominant with five EMADE non-dominated solutions. The incorporation of DNN architectures as primitives in EMADE remains an avenue for future work. In Figure 4.37, the green points were hand-crafted algorithms based on computer vision techniques used to seed the optimization. Note that almost all of these techniques are dominated (outperformed in both dimensions) by EMADE solutions.

Figure 4.38 shows one of the EMADE evolved solutions that was co-dominant with the DNN. The evolved solution outperformed the DNN on false positive rate, while underperforming on true positive rate.

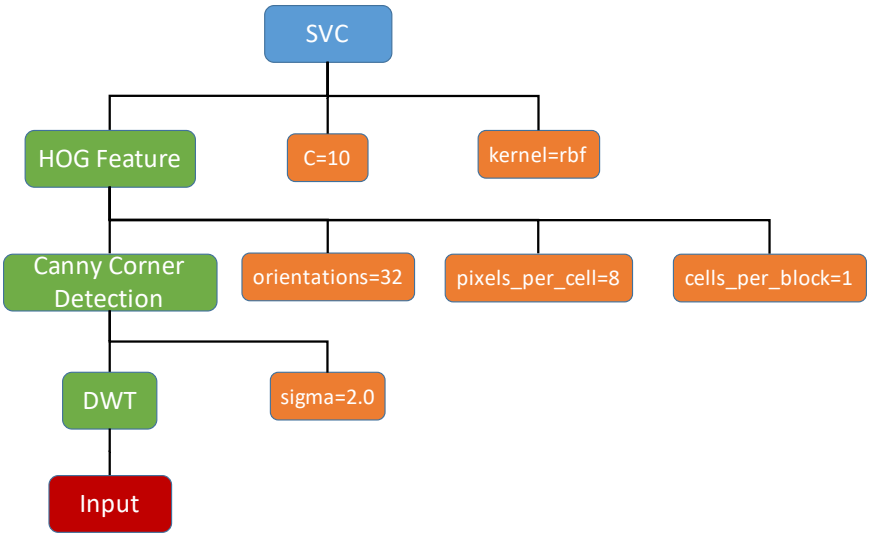


Figure 4.38: An evolved solution produced by EMADE with false positive rate of 0.0285 and false negative rate of 0.0575

4.6.5 Conclusions

This work demonstrates EMADE's capabilities on the important domain of image processing for ISR applications. We showed the ability to combine and optimize high-level, state-of-the-art techniques into new algorithms.

We improved EMADE by creating new primitives, developing seed algorithms, and making changes to the infrastructure to support image data and increase throughput by caching data on disk. Future work could look at how to better implement deep neural networks into EMADE as primitives that can be trained during the evolutionary process. One could also identify how to optimize not only the hyperparameters, but the architectures (i.e. layers and configurations) of the deep neural networks as well.

CHAPTER 5

EXPLORATION OF EVOLUTIONARY OPERATORS

Chapter 4 showed that EMADE can solve machine learning challenges across diverse domains. However, EMADE required a significant amount of time between generations. Figures 4.4 and 4.32 showed that this time can easily exceed an hour, even on a simple problem such as the Titanic dataset.

The main focus of this chapter is to decrease the time it takes to run EMADE, namely, converge to better solutions more quickly. Here, we define a lower area under the curve of the Pareto frontier (for a minimization problem in two dimensions) of an evolutionary process to be a "better solution." In higher dimensions, this is a reduction of area under a hypervolume. We define "More quickly" as fewer individuals requiring training and evaluation. To achieve these performance improvements, we are willing to suffer additional computational time in the evolutionary loop, including selection, mating, and mutation,in exchange for a higher probability of producing individuals that contribute more significantly to evolutionary progress. We are willing to sacrifice some performance in the main loop because evaluations of the high-level primitives used in EMADE are far more computationally intensive than traditional genetic programming. For example, evaluating on the five folds of the 891 instances in the Titanic feature set took an average time of about 3.5 minutes.

To achieve this improved performance, we investigate two concepts in the evolutionary process: fuzzy selection and matchmaking. Fuzzy selection focuses on retaining more statistical information in the selection process than traditional high-level objective scores. Matchmaking focuses on a probabilistic method of pairing parents to generate more productive children.

5.1 Fuzzy Selection

5.1.1 Introduction

In the current age of big data, a new field of data science has been emerging, seeking methods to automate the process of building machine learning algorithms through using multiple-objective vector-based genetic programming [72, 73] or machine learning pipeline optimization [74, 75]. There are two significant differences between these methods and traditional genetic programming. First, evaluations are extremely expensive when compared to that of a scalar-based tree structure. Second, our objectives are all based on aggregate statistics, such as true positive or false positive rates, computed from multitudes of test cases.

While Chapter 4 showed that traditional selection methods such as NSGA2 or SPEA2 perform acceptably in evolving machine learning algorithms from these aggregate statistics, these selection methods have been designed to be quick and efficient. In traditional genetic programming, the selection algorithm is often the bottleneck of the evolutionary process. However in our case, the evaluations take most of the processing time. We seek to find selection methods that may take more processing time, but also take advantage of the underlying statistical behavior of the machine learning algorithm to produce better solutions (measured by area under the Pareto curve) more quickly (measured in number of individuals evaluated).

This section details the concept of fuzzy selection, which focuses on estimating the statistics of each solution in objective space from test cases, rather than a single point. Using these statistical representations allows for dominance comparisons in terms of probability rather than a resolved state. For example, in a pairwise comparison using scalar objectives, one individual would either be dominant to, dominated by, or co-dominant with respect to the other. In a fuzzy paradigm, all of these cases have some probability of occurring, based upon the distributions of the underlying test cases used to compute each

objective score. We propose a method for computing a probability of selection based upon a probability of Pareto-dominance, rather than using a resolved state of dominance between coordinates in objective space. We estimate this probability through covariance estimation combined with Markov chain Monte Carlo simulation. This probability is then used with a roulette wheel selection method to directly tie the probability of Pareto-dominance to the probability of selection. We will compare performance to the traditional selection methods NSGA2 and SPEA2.

5.1.2 Related Work

Traditional Multi-Objective Selection Methods

One major component of any genetic algorithm is the selection of parent genomes to produce the next generation of individuals. In multi-objective optimization, most of these algorithms use Pareto-optimality. Equation 1.1 showed the definition of Pareto-optimality. Section 2.3 summarized the various common selection techniques that use Pareto-optimiallity, including SPEA, SPEA2, NSGA-II, and PESA.

While methods based on Pareto-optimality have been shown to be successful, we conjecture there is more to be gained by exploring the underlying statistics of the scalar objectives. For example, if on a particular objective of a binary classifier there are 100 test cases used to compute a probability of detection, there are "100 choose 50" (more than $1 \cdot 10^{29}$) different individuals that give a 50% error rate. By collapsing these test cases to their averages to use traditional selection techniques, we lose information that could potentially help steer the population to new and interesting solutions.

Test Case based problems

Traditionally, evolutionary machine learning is performed on aggregate scores of large numbers of test cases. In recent years, focus has shifted toward vectorized information on how a classifier or predictor performs on each test case. Some of these methods in-

clude novelty search [14], discovery of search objectives by clustering [76], and discovery of search objectives by non-negative matrix factorization [77]. The latter two of these approaches derive a small number of objectives from a large number of test cases via an interaction matrix. The interaction matrix places each individual on a row, and the result from each test case across the columns. They then use the traditional multi-objective approaches such as those described in the previous section on the derived objectives. While these methods produce good results and isolate unique individuals, we believe they are still limited by not capturing the statistics of the underlying objectives.

Novelty search removes the sense of objectives entirely, selecting based on the uniqueness of the solution in behavioral space. For purposes of a binary classifier, this would be the vector containing the predictions for each test case. The higher the distance between this vector and the rest of the population, the higher the probability of selection. Novelty search rewards this distance to increase phenotypic diversity. While this method more freely explores the behavior space, that does not guarantee the solution will quickly converge to something that is objectively good. In fact, the search itself is, by design, divergent.

5.1.3 Motivation

Probability of Dominance

There are many selection techniques like NSGA2, SPEA2, MOEA/D. Each can be considered a black box that takes in a population, and using some probabilistic method, returns a new population. In effect, all of these methods have control over the probability of selection per individual. In single-objective optimizations using a scalar objective score, such as accuracy, a simple roulette selection provides a simple mapping from objective score to probability of selection. In multi-objective methods, we desire a probability of selection that can be fed into a roulette wheel based on dominance relationships and crowding. Instead, most multi-objective selection techniques rely on Pareto rankings and strengths combined with sorting or tournaments.

For real-world applications of evolved machine learning, we have scalar objectives (e.g. an average error rate) computed from many observations that are fed to the machine learning algorithm as test cases. Despite all of the statistical information that can be derived from these performance vectors, we traditionally only use the aggregate scalar objective score. Instead, we can compute statistical information for each objective based on the performance distribution of test cases, and compute a probability of dominance rather than a resolved state based on the scalar.

We would like to feed "probabilities of dominance" into a roulette wheel. Given M objectives and N individuals, the probability that X_i is non-dominated in a minimization scheme is one minus the probability that it is dominated, which is the probability that there is some individual X_j that is lower on each of the M objectives of individual X_i. Summed up mathematically, the probability X_i is non-dominated is

$$P(X_i \prec X_{j=1\ldots N, j\neq i}) = 1 - \bigcup_{j\in 1\ldots N, j\neq i} \bigcap_{m=1}^{M} P(X_j(m) < X_i(m)).$$

We refer to this as the probability of Pareto-dominance, or probability of dominance for short. Unfortunately, many challenges exist when computing this probability. Given a population of individuals, we do not have independence on the performance of test cases, performance on aggregate scalar objectives, or performance on individuals. If this independence were present, computing the probability of dominance would be a straightforward multiplication. Since we do not have independence, rather than directly computing this quantity, we use a Markov chain Monte Carlo approach to simulate a probability of dominance that drives selection.

5.1.4 Methods

Computing Covariance

To estimate a probability of dominance using Monte Carlo, we need to simulate potential solutions in objective space from a population. Taking each objective per individual to be a random variable, we use the result from each test case of an individual to represent an observation on that random variable. We can construct an NM by K observation matrix, where N is the number of individuals, M is the number of objectives, and K is the number of test cases. In practice, however, it is rare to find a problem with an equal number of test cases for each objective. For example, in a binary classification problem, we may have more test cases for false positive error than we do for false negative error. To rectify this, we group the test cases such that each objective has the same number of test case groups, allowing us to form one-to-one relationships between our random variables. For each group, the result is the average of the test cases in that group. We choose the number of groups to be the minimum number of test cases that make up each objective; as a result, at least one objective will have one test case per test case group. While these groups lose some resolution on test case correlations, they offer significantly more information than the higher-level aggregate objectives.

Once the observation matrix $\boldsymbol{X}$ has been constructed, we use it to compute the covariance matrix $\boldsymbol{\Sigma}$ between our NM variables, resulting in an NM by NM matrix. Ideally, this covariance matrix would be

$$\boldsymbol{\Sigma}_{ij} = \frac{1}{K}\sum_{k=1}^{K}(\boldsymbol{X}_{ik} - E[\boldsymbol{X}_i])(\boldsymbol{X}_{jk} - E[\boldsymbol{X}_j]).$$

However, an issue arises with the number of objectives and individuals typically found in a multi-objective population based algorithm, where large covariance matrices become ill conditioned. These covariance matrices are needed in the Markov chain Monte Carlo simulation, but are not directly usable due to numerical precision issues in computing their

inverses. To solve this issue, we use the Ledoit-Wolf method [78] as implemented in scikit-learn [37], which results in a well-conditioned, invertible estimate of the covariance matrix.

Markov Chain Monte Carlo

Denote the estimate of the covariance matrix as $\boldsymbol{\Sigma}$, and the expected value for each objective for each individual as

$$\boldsymbol{\mu} = E\left[\boldsymbol{X}_1\right], E\left[\boldsymbol{X}_2\right], \ldots, E\left[\boldsymbol{X}_{NM}\right].$$

We now draw samples from the distribution they represent. Following our binary classifier example we use a truncated multivariate normal distribution on the range of [0,1]. This is because false negative and false positive rates cannot fall outside those bounds. We represent our distribution as

$$\mathcal{TN}(\boldsymbol{\mu}, \boldsymbol{\Sigma}).$$

Drawing samples from a distribution can be as simple as drawing a random sample from a uniform distribution on [0,1] and using it to sample the desired distribution's cumulitive distribution function (CDF). However, because a truncated multivariate normal distribution has no closed-form solution for its CDF, we instead implement a Gibbs sampler prescribed by Wilhelm [79].

To describe Wilhelm's sampler, let j be the Monte Carlo trial number and i be our position in the NM dimensional vector $\boldsymbol{x}$. We draw a sample

$$\begin{aligned} &x_{i.-i} = \mu_{i.-i} + \\ &\sigma_{i.-i}\Phi^{-1}\left[U\left(\Phi\left(\frac{1-\mu_{i.-i}}{\sigma_{i.-i}}\right) - \Phi\left(\frac{0-\mu_{i.-i}}{\sigma_{i.-i}}\right)\right) + \Phi\left(\frac{0-\mu_{i.-i}}{\sigma_{i.-i}}\right)\right], \end{aligned}$$

where the notation $i. - i$ means sample i given all samples besides i, U randomly drawn

from $Uni(0,1)$, and Φ is the CDF of a standard normal $\mathcal{N}(0,1)$. Furthermore,

$$\mu_{i.-i} = \mu_i - H_{ii}^{-1} H_{i,-i} \left(\boldsymbol{x}_{-i} - \boldsymbol{\mu}_{-i} \right),$$

and

$$\sigma_{i.-i} = \sqrt{H_{ii}^{-1}}.$$

H is the precision matrix

$$H = \Sigma^{-1}.$$

Finally,

$$\boldsymbol{x}_{-i} = x_1^{(j)}, \ldots, x_{i-1}^{(j)}, x_{i+1}^{(j-1)}, \ldots, x_{NM}^{(j-1)},$$

and

$$\boldsymbol{\mu}_{-i} = \mu_1^{(j)}, \ldots, \mu_{i-1}^{(j)}, \mu_{i+1}^{(j-1)}, \ldots, \mu_{NM}^{(j-1)}.$$

This sampling is the slowest part of this proposed selection algorithm by several orders of magnitude, and scales with the number of objectives, individuals, and trials. The time to perform the sampling is not dependent on the number of test cases, which would impact the evaluation time of an individual's objective scores.

Now that we have J samples of NM variables, we use this matrix to compute a probability of non-dominance. We can iterate through each NM dimensional sample and deinterlace it to form N vectors of M elements each. Each of these N vectors represents a possible location in objective space for an individual given by the M elements. Because we draw the vector simultaneously, the correlations between the individuals and their objective scores are respected. We compute the non-dominated set of the N individuals for each sample, and maintain a 1 by N histogram of the number of times each individual was found to be non-dominated throughout the J samples. This histogram is then normalized such that the sum is 1. The value N_i is the probability that individual i is a non-dominated solution. These probabilities drive a roulette wheel selection with replacement, such that

the values in the histogram truly represent a probability of selection.

5.1.5 Experiment

To test our proposed fuzzy selection technique, we work a binary classification problem from the Physical Activity Monitoring Data Set (PAMAP2) [80]. This data was collected from nine subjects, each wearing three inertial measurement units and a heart rate monitor while performing 12 different activities. Prior to feature construction, we cleaned the time series data from the sensors as prescribed by Baldominos et al. [81]:

1. A boolean mask for the 54 (Indexed [0-53]) time series sensor data columns is created.
   (a) Column 0 is marked for removal, corresponding to the time stamp of the data. We remove this column because we do not want to train on the time of an activity.
   (b) Columns 16-19, 33-36, and 50-53, which represent orientation data that should not be used as features, are marked for removal as indicated in [80].
2. Remove all columns marked for removal above in step 1.
3. Due to differing sampling rates of the sensors, there are missing data points in the data. Fill all the missing data by propagating samples forward.
4. At this point, there may be some leading data points that are missing. Fill these by propagating backward.
5. Because we do not know what behaviors were occurring during transient activities (rows with class label 0), we do not want to train on or classify these behaviors. Remove all rows with class label 0.

Once the data has been cleaned, features are constructed from the 40 remaining time series columns as prescribed by Baldominos [81]:

1. Each column is transformed using a sliding fast Fourier transform (FFT) of size 512 samples, which represents 5.12 seconds of 100 Hz data.

2. From each of set of the 512 coefficients (257 real, 255 imaginary broken out from the first half of the FFT), 7 statistical features are extracted: mean, median, standard deviation, max, min, and first and third quartiles.

3. The resulting feature matrix is N windows by 280 features (7 statistics on each of the 40 columns).

We step by 6.25% (1/16th of the window, or 0.32 seconds) to generate our feature matrix. We experimented with different step sizes and found them to have little impact on the results.

Given the feature matrix, we select a single activity to classify from the 280 features. For the example, if we select the activity of vacuuming, which is class label 16 in the data set, all instances of class 16 are replaced with a 1, while all other instances' class labels are replaced with a 0. For vacuuming, this produces 701 true positive samples, and 6,888 false positive samples. This unbalanced data set is an excellent candidate for multi-objective optimization, where driving by accuracy alone would quickly favor suppressing false positives. We repeat this partitioning on each of the eight activities carried out by all subjects: lying, sitting, standing, walking, ascending stairs, descending stairs, vacuuming, and ironing.

The goal is to understand how using the statistical nature of the underlying test cases (of which there are 7,589 for vacuuming) to compute a probability of dominance compares with traditional multi-objective approaches that use only the false negative and false positive error rates. We compare with a linear boolean genome of length 280, which serves as a selection mask for our feature matrix prior to training a naive Bayes classifier. We chose this machine learning method for its quick training and evaluation time to run through generations more quickly, in addition to its deterministic results when implemented in scikit-

learn.

Most evolutionary settings we used matched those used by Baldominos [81]. We added elitism to the optimization to push the optimization along faster. The evolutionary settings were:

- Population size 512
- Single-point crossover rate 35%
- Mutation bit flip
  - 8.333% per individual
  - 8.333% per gene in the individual
- Elitism: Every generation selection is performed from the Pareto frontier $+$ current offspring
- Genome: 280 boolean attributes
- Fuzzy Pareto method uses $10,000$ Monte Carlo trials

5.1.6 Results

Comparison to Traditional Methods

To analyze the performance of the fuzzy selection method, three optimizations were created, where the only variable was the selection method used, being the proposed fuzzy selection, NSGA2, and SPEA2. Results of running this optimization are shown in Figure 5.1 for the binary classifier of the vacuuming activity. The top plot compares the proposed fuzzy selection with SPEA2, while the bottom shows the same fuzzy selection against NSGA2. In both cases, fuzzy selection outperforms both traditional methods on the average case, best case, and worst case. We repeated the experiment for the seven other

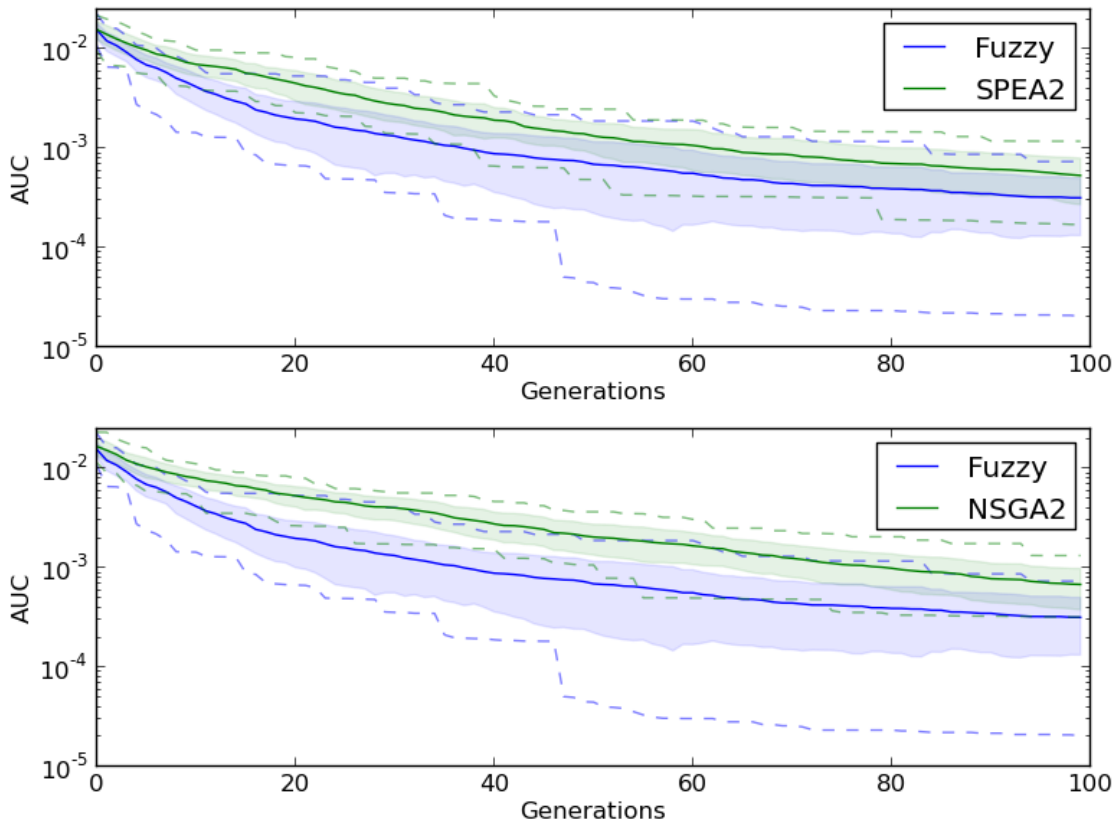


Figure 5.1: Average area under the curve (AUC) of the Pareto frontier of 30 trials of 100 generations during the evolution of a binary classifier for vacuuming. Shading shows $\pm$ one standard deviation. Dashed lines show max and mins. AUC is shown on a log scale to emphasize the later generations.

Table 5.1: Results on 30 trials of 100 generations optimizing a binary activity classifier. Note that μ and σ of AUC are computed over the trials, and the subscripts represent generations. Best and worst represent the minimum and maximum over the trials.

Activity	Method	μ_0	μ_{99}	σ_{99}	$\mu_{\text{gen}} \leq \frac{1}{2}\mu_0$	Best_{99}	Worst_{99}
Lying	Fuzzy	$8.66 \cdot 10^{-2}$	$\mathbf{4.87 \cdot 10^{-2}}$	$2.98 \cdot 10^{-3}$	≥ 100	$\mathbf{3.62 \cdot 10^{-2}}$	$\mathbf{5.28 \cdot 10^{-2}}$
	SPEA2	$8.84 \cdot 10^{-2}$	$5.29 \cdot 10^{-2}$	$\mathbf{1.32 \cdot 10^{-3}}$	≥ 100	$5.04 \cdot 10^{-2}$	$5.76 \cdot 10^{-2}$
	NSGA2	$8.77 \cdot 10^{-2}$	$5.32 \cdot 10^{-2}$	$1.33 \cdot 10^{-3}$	≥ 100	$5.16 \cdot 10^{-2}$	$5.76 \cdot 10^{-2}$
Sitting	Fuzzy	$4.55 \cdot 10^{-1}$	$2.72 \cdot 10^{-1}$	$9.92 \cdot 10^{-2}$	≥ 100	$1.19 \cdot 10^{-1}$	$3.51 \cdot 10^{-1}$
	SPEA2	$4.54 \cdot 10^{-1}$	$2.34 \cdot 10^{-1}$	$\mathbf{9.63 \cdot 10^{-2}}$	≥ 100	$\mathbf{9.54 \cdot 10^{-2}}$	$\mathbf{3.28 \cdot 10^{-1}}$
	NSGA2	$4.47 \cdot 10^{-1}$	$\mathbf{2.25 \cdot 10^{-1}}$	$1.01 \cdot 10^{-1}$	≥ 100	$1.10 \cdot 10^{-1}$	$3.35 \cdot 10^{-1}$
Standing	Fuzzy	$8.54 \cdot 10^{-2}$	$5.39 \cdot 10^{-3}$	$7.00 \cdot 10^{-3}$	14	$1.18 \cdot 10^{-3}$	$3.65 \cdot 10^{-2}$
	SPEA2	$8.43 \cdot 10^{-2}$	$\mathbf{1.61 \cdot 10^{-3}}$	$2.74 \cdot 10^{-3}$	**13**	$\mathbf{6.25 \cdot 10^{-4}}$	$1.63 \cdot 10^{-2}$
	NSGA2	$8.37 \cdot 10^{-2}$	$2.62 \cdot 10^{-3}$	$\mathbf{2.64 \cdot 10^{-3}}$	18	$9.50 \cdot 10^{-4}$	$\mathbf{1.49 \cdot 10^{-2}}$
Walking	Fuzzy	$8.51 \cdot 10^{-2}$	$5.50 \cdot 10^{-2}$	$7.82 \cdot 10^{-3}$	≥ 100	$4.31 \cdot 10^{-2}$	$7.24 \cdot 10^{-2}$
	SPEA2	$8.77 \cdot 10^{-2}$	$3.77 \cdot 10^{-2}$	$\mathbf{1.50 \cdot 10^{-3}}$	**39**	$3.54 \cdot 10^{-2}$	$3.98 \cdot 10^{-2}$
	NSGA2	$8.37 \cdot 10^{-2}$	$\mathbf{3.70 \cdot 10^{-2}}$	$1.84 \cdot 10^{-3}$	48	$\mathbf{3.25 \cdot 10^{-2}}$	$\mathbf{3.98 \cdot 10^{-2}}$
Ascending Stairs	Fuzzy	$1.15 \cdot 10^{-1}$	$\mathbf{4.98 \cdot 10^{-2}}$	$\mathbf{2.23 \cdot 10^{-3}}$	**35**	$\mathbf{4.35 \cdot 10^{-2}}$	$\mathbf{5.28 \cdot 10^{-2}}$
	SPEA2	$1.14 \cdot 10^{-1}$	$5.49 \cdot 10^{-2}$	$4.59 \cdot 10^{-3}$	91	$4.62 \cdot 10^{-2}$	$6.43 \cdot 10^{-2}$
	NSGA2	$1.15 \cdot 10^{-1}$	$5.90 \cdot 10^{-2}$	$5.19 \cdot 10^{-3}$	≥ 100	$4.97 \cdot 10^{-2}$	$6.93 \cdot 10^{-2}$
Descending Stairs	Fuzzy	$1.42 \cdot 10^{-1}$	$\mathbf{6.23 \cdot 10^{-3}}$	$\mathbf{7.14 \cdot 10^{-3}}$	**12**	$\mathbf{3.24 \cdot 10^{-3}}$	$\mathbf{3.70 \cdot 10^{-2}}$
	SPEA2	$1.43 \cdot 10^{-1}$	$1.96 \cdot 10^{-2}$	$8.78 \cdot 10^{-3}$	35	$5.32 \cdot 10^{-3}$	$4.09 \cdot 10^{-2}$
	NSGA2	$1.44 \cdot 10^{-1}$	$2.10 \cdot 10^{-2}$	$7.29 \cdot 10^{-3}$	41	$1.26 \cdot 10^{-2}$	$4.22 \cdot 10^{-2}$
Vacuuming	Fuzzy	$1.58 \cdot 10^{-2}$	$\mathbf{3.22 \cdot 10^{-4}}$	$\mathbf{1.87 \cdot 10^{-4}}$	**5**	$\mathbf{2.10 \cdot 10^{-5}}$	$\mathbf{7.45 \cdot 10^{-4}}$
	SPEA2	$1.59 \cdot 10^{-2}$	$5.40 \cdot 10^{-4}$	$2.65 \cdot 10^{-4}$	9	$1.73 \cdot 10^{-4}$	$1.19 \cdot 10^{-3}$
	NSGA2	$1.69 \cdot 10^{-2}$	$6.90 \cdot 10^{-4}$	$3.07 \cdot 10^{-4}$	9	$3.24 \cdot 10^{-4}$	$1.35 \cdot 10^{-3}$
Ironing	Fuzzy	$2.25 \cdot 10^{-2}$	$\mathbf{2.37 \cdot 10^{-3}}$	$\mathbf{2.04 \cdot 10^{-4}}$	**8**	$\mathbf{1.89 \cdot 10^{-3}}$	$\mathbf{2.78 \cdot 10^{-3}}$
	SPEA2	$2.31 \cdot 10^{-2}$	$3.13 \cdot 10^{-3}$	$5.02 \cdot 10^{-4}$	19	$2.21 \cdot 10^{-3}$	$4.37 \cdot 10^{-3}$
	NSGA2	$2.21 \cdot 10^{-2}$	$3.30 \cdot 10^{-3}$	$6.32 \cdot 10^{-4}$	25	$2.51 \cdot 10^{-3}$	$4.81 \cdot 10^{-3}$

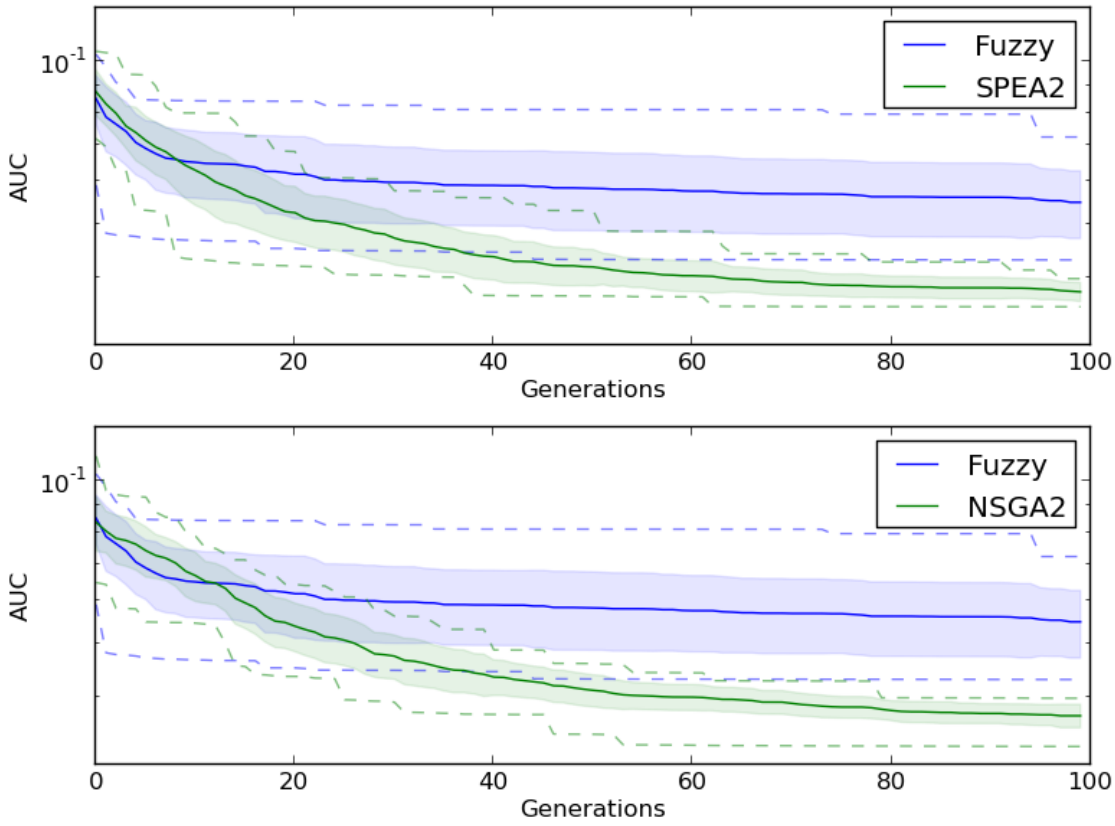


Figure 5.2: Average area under the curve (AUC) of the Pareto frontier of 30 trials of 100 generations during the evolution of a binary classifier for walking. Shading shows $\pm$ one standard deviation. Dashed lines show max and mins. AUC is shown on a log scale to emphasize the later generations.

activities in the protocol from PAMAP2: lying, sitting, standing, walking, ascending stairs, descending stairs, and ironing. Results of all experiments are shown in Table 5.1.

Although fuzzy selection outperforms the two standard methods for ascending stairs, descending stairs, vacuuming, and ironing on all measured metrics, fuzzy selection is not a silver bullet. Fuzzy selection underperforms on some or all metrics for lying, sitting, standing, and walking. To illustrate some of the indicator characteristics that may suggest fuzzy selection will perform well, we compare vacuuming, where fuzzy selection outperformed on all metrics, and walking, where fuzzy selection underperformed on all metrics.

Figure 5.2 shows the 30 trial average area under the curve of the Pareto frontier as a function of generation number for the evolution of the walking classifier. This evolution

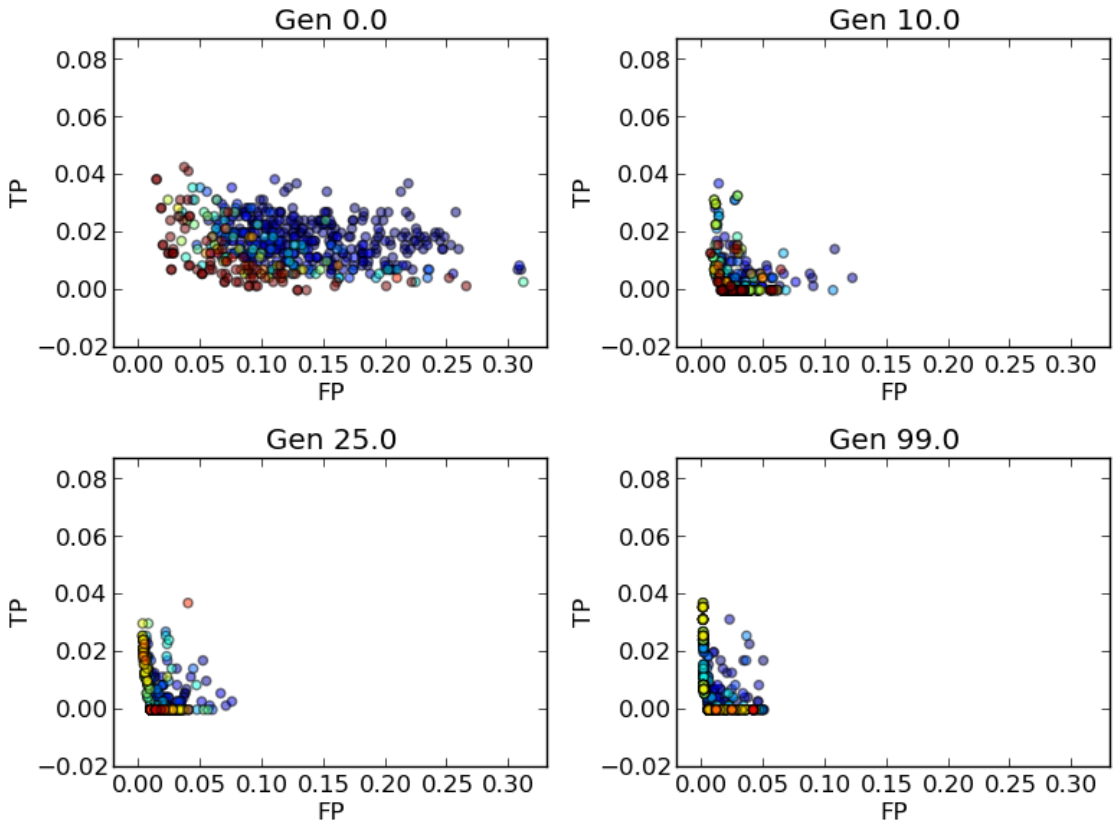


Figure 5.3: Four generation snapshots into the evolution of a vacuuming classifier. Each point plots the average true positive error rate vs. false positive rate for a genome. Color shows the expected number of selections for the next generation per genome from blue (0 selections) to red (more than 2 selections).

behaves differently than that of the vacuuming classifier shown previously in Figure 5.1. The fuzzy selection for the walking classifier, while starting out faster than SPEA2 and NSGA2, quickly levels out around generation 10, in contrast to continuously decreasing as it did in the vacuuming evolution. However, SPEA2 and NSGA2 both continue to decrease for the walking classifier. This highlights some systematic issue in the fuzzy selection method.

Population Simulation

To better understand the successes and failures of fuzzy selection, we look at the behavior of fuzzy selection over the course of the evolution. Figure 5.3 shows scatter plots of objective

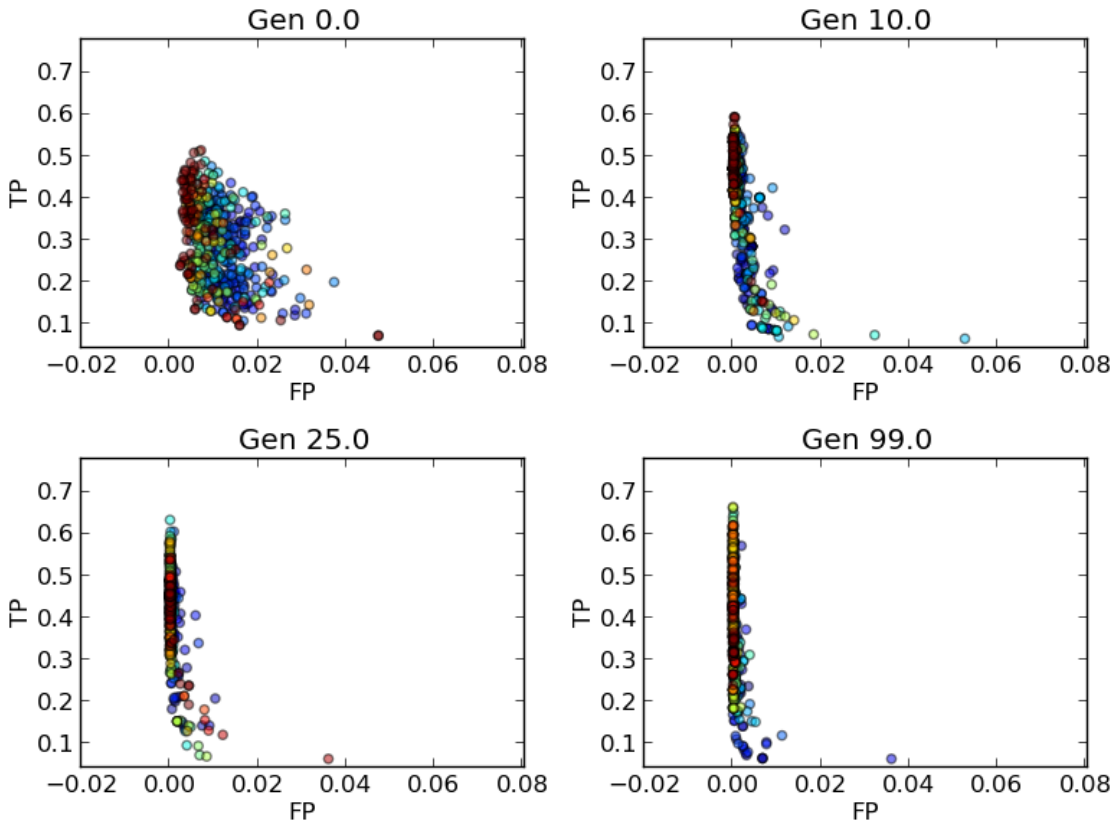


Figure 5.4: Four generation snapshots into the evolution of a walking classifier. Each point plots the average true positive error rate vs. false positive rate for a genome. Color shows the expected number of selections for the next generation per genome from blue (0 selections) to red (more than 2 selections).

space for four of the 100 generations evaluated for the vacuuming classifier, while Figure 5.4 shows the same generational snapshots for the walking classifier. The y-axis of each plot shows the false negative rate, i.e., the probability the classifier predicted a vacuuming sample as not vacuuming. The x-axis of each plot shows the false positive rate, i.e., the probability the classifier predicted a non-vacuuming sample as vacuuming. Each point on the plot represents the expected error rates for each of the two objectives computed over all test cases. The color of each point illustrates the individual's expected number of selections per generation, ranging from blue (zero selections) to red (more than two selections).

Starting at generation zero, the initial results of the random population are evaluated, and the probability of selection draws towards the origin as expected. This is true for both the vacuuming classifier and the walking classifier. By generation 10, the distribution of individuals in objective space begins to form an L, and probability of selection is highest for the points closest to the edges. This trend continues through the evolution as shown in generation 25 and the final generation, 99. The differences between the vacuuming and walking evolutions become apparent in the plot for generation 25. At this point, the vacuuming classifier still has a high probability of selection near the origin, but the walking classifier has a lower probability of selection there than it does rising up the axis corresponding to false negative error rates.

A benefit of fuzzy selection is its implicit handling of crowding. If there are many solutions in one section of objective space, they naturally split the probability of dominance between them, especially if those solutions are highly correlated. On the other hand, solutions that do not have many neighbors are be more likely to be Pareto-optimal when they generate strong results from their random trials. This is observed in the scatter plots; points far behind the Pareto frontier often have a stronger probability of selection than those that are closer to the frontier, but also have more neighbors.

Some regions in objective space are crowded, but have particular individuals with high expected selections. These are individuals that capture unique solutions relative to their

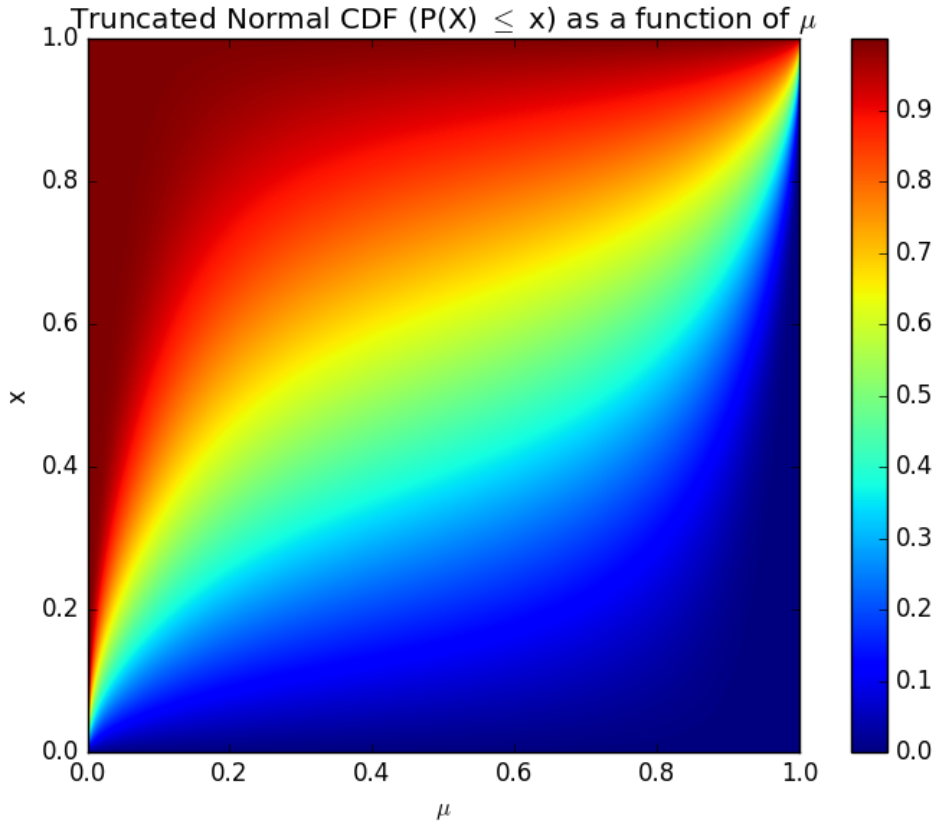


Figure 5.5: Visualization of cumulative distribution function of a truncated normal in a single dimension from an underlying Bernoulli random variable.

neighbors, i.e. their means are similar on both objectives, but their performance on the underlying test cases is not highly correlated.

Due to higher variances of the underlying test cases, individuals along the axes may produce similar probabilities of selection to those that are closer to the origin. This is due to the underlying Bernoulli random variable, where the variance for each objective is

$$\sigma^2 = p(1-p),$$

where p is the probability the event occurred. This is maximized when an objective is at 0.5 and falls off as p increases or decreases.

Figure 5.5 visualizes a cumulative distribution function for a one-dimensional truncated

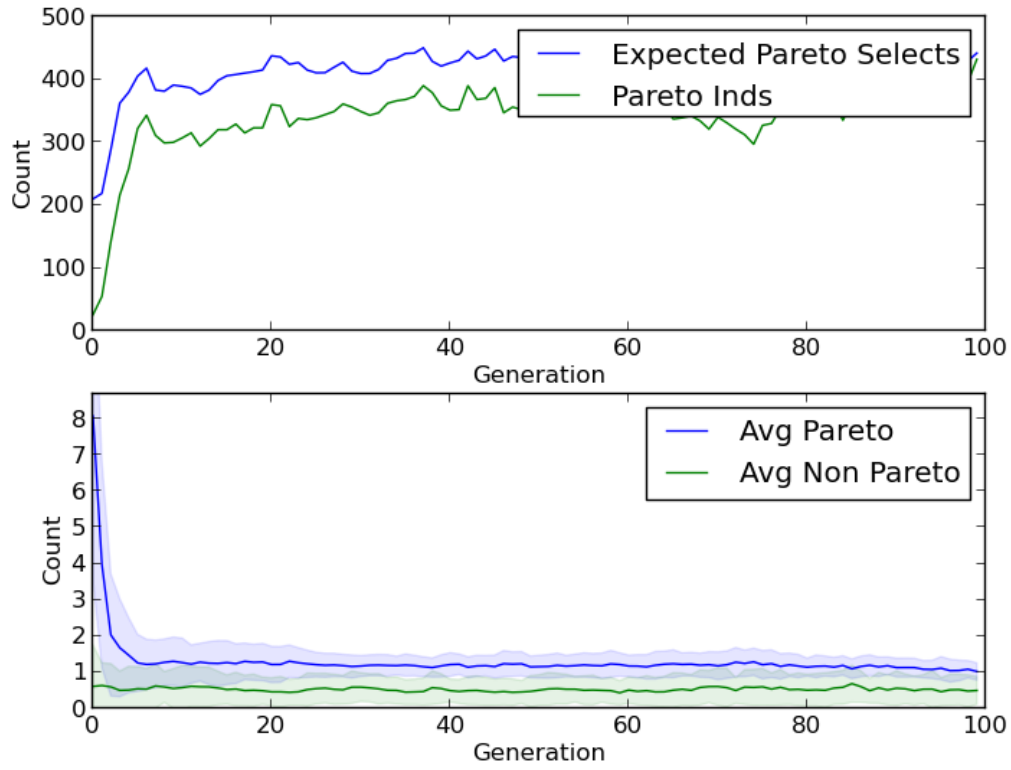


Figure 5.6: Expected number of selections, and average number of selections, per generation of Pareto and non-Pareto individuals during the evolution of a binary classifier for vacuuming.

normal distribution with $\mu = p$ and $\sigma^2 = p(1-p)$. The color shows the value of the CDF for each possible value of μ across the bounded region $0 \leq x \leq 1$. As μ increases, the horizontal band representing the 10^{th} percentile (i.e., x=0.1) has a slow rate of increase in $P(x) \leq x$. For instance, an individual with an average objective score of $\mu = 0.4$ has a similar probability to produce a solution of $x < .1$ as an individual with an objective score of $\mu = 0.2$. Individuals that are significantly closer to the origin, are not necessarily more likely to produce non-dominated individuals, especially when factoring in multiple objectives.

Consider the expected number of selections per generation of Pareto and non-Pareto individuals (where the Pareto optimality of an individual is determined by its average ob-

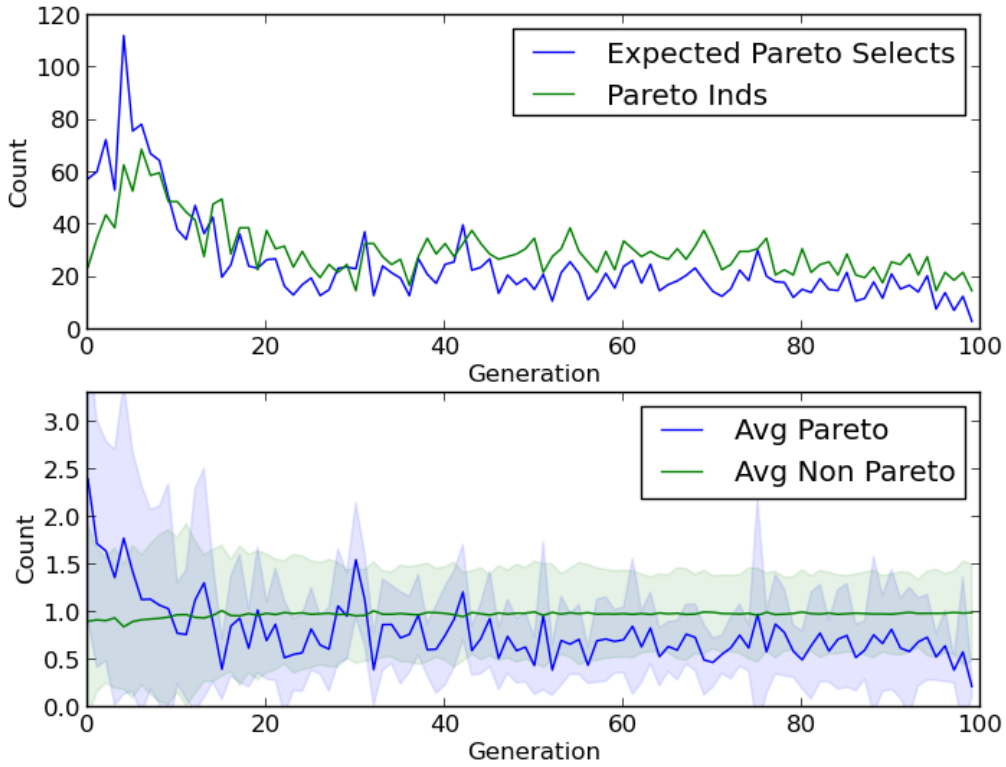


Figure 5.7: Expected number of selections, and average number of selections, per generation of Pareto and non-Pareto individuals during the evolution of a binary classifier for walking.

jective score across all test cases). Figure 5.6 shows that in the evolution of a vacuuming classifier, fuzzy selection is appropriately favoring Pareto individuals, selecting Pareto individuals on average twice as likely than non-Pareto individuals. The favoritism is higher in earlier generations when the Pareto solutions are larger drivers, but as the population tightens towards the end, the ratio begins to decline. This is due to a number of factors, including the increase of Pareto individuals in the population over time. Both the number of Pareto individuals and the expected number of selections of Pareto individuals grow over time, leveling off at about 80% of the population size of 512. There are always more expected selections of Pareto individuals than there are Pareto individuals, which implies Pareto individuals are being selected more than once each. This contrasts with the performance of the walking classifier shown in Figure 5.7. Here, the number of Pareto individuals and expected Pareto selections have a much noisier characteristic over the course of the optimization. The average number of non-Pareto selections crosses that of Pareto selections at about the same generation the area under the Pareto front starts to level off in Figure 5.2. The expected number of Pareto individuals selected in the walking classifier evolution levels off at approximately 6% of the population, in contrast with the 80% expected in the evolution of the vacuuming classifier. For this dataset, favoritism of Pareto selections to non-Pareto selections is an indicator as to whether or not fuzzy selection will perform successfully.

5.1.7 Conclusions

Fuzzy selection can offer significant improvements over traditional multi-objective approaches by taking into consideration the statistical nature of the objective performance. In our relative best case, which was the classifier for ascending stairs, we achieved a reduction of 68.21% in final area under the curve from SPEA2. We also showed a speedup of 291.66% in reaching the generation that halves the initial area under the curve.

However, fuzzy selection is also sensitive to the underlying statistics of test cases, and

in some cases underperforms when compared to those same traditional approaches. In our relative worst case, optimizing a standing classifier, we converged to a final area under the curve that was an increase of 234.78% over SPEA2, but only suffered a slowdown of 7.69% to reach the generation that halves the initial area under the curve.

Fuzzy selection takes several orders of magnitude longer to perform for a $10,000$ trial, two objective, 512 individual population than the traditional selection methods. This is by design, as the selection process still takes less than one typical evaluation of a reasonably challenging machine learning data set. For smaller populations, fewer trials are needed, and we can achieve faster performance.

Although we have found indicators as to when fuzzy selection may perform successfully, such as favoring the selection of Pareto individuals over non-Pareto individuals, further work is necessary to understand the underlying mechanism that causes fuzzy selection to outperform or underperform traditional multi-objective techniques. Ideally, one would look for a set of problem features that is well suited to fuzzy selection, and a systematic method to correct the fuzzy selection procedure to improve performance on problems that are currently not well suited for this method.

Finally, we would like to demonstrate success on additional problems, including the benchmarking suite used to analyze the performance of the non-negative matrix factorization technique by Liskowski and Krawiec. We would like to compare the performance of a corrected fuzzy selection method against various test case based methods including: the NMF search drivers, novelty search, and online discovery of search objectives (DISCO).

5.2 Crossover Improvement

5.2.1 Introduction

Because evaluations of individuals is expensive, we want to achieve our best solutions in the fewest number of evaluations. To this end, we seek evolutionary operations that can be used to improve performance. In this section, we analyze the pairing of parents before

crossover. The current standard procedure following selection in conventional algorithms (such as SPEA2[12] and NSGA2[9]) is a random choice of parents.

Some newer algorithms introduce crossover restriction to increase performance through techniques such as locality (as in MOEA/D[15]) or dominance restrictions (as in [82]). We think of these techniques as point solutions in the full space of pairing algorithms we wish to investigate.

We take a more abstract approach and assume that we can design a matchmaking algorithm, whose role is to take a list of parents and return unique pairs that should produce the highest probability of generating successful offspring. Assuming this hypothetical matchmaker exists, this section asks: what sort of increase of the probability of generating successful offspring is required to affect a statistically significant performance increase in the evolutionary process?

The first part of this section deals with methods for comparing evolutionary algorithms. Next, we define two different notions of what makes a crossover successful or unsuccessful. We then outline some expectations and present a simulated approach to the evolutionary process, which functions without a genome. We apply this simulation to understand how the probability of successful crossover impacts the area under the non-dominated frontier. Finally, we use these results as an aid in designing experiments for a potential matchmaking algorithm.

5.2.2 Comparing Results of Evolutionary Algorithms

We require a method of comparing two evolutionary processes, a difficult task for stochastic, multiple-objective problems. There has been a significant amount of research into the comparison of evolutionary methods, both qualitative and quantitative.

For qualitative methods, Zitzler et al. [83] performed a comparative case study of multi-objective evolutionary algorithms. They focused their efforts on understanding the types of solutions that could be reached by each algorithm and reported results based on pairwise

comparisons of final frontiers, rather than looking at an area under the curve. Zitzler et al. [84] later looked at many different metrics for comparing algorithm performance on multiple objectives qualitatively, but did not use statistical tests to empirically show that one performs better than another.

Quantitative methods use repeated trials to capture the stochastic nature of evolutionary algorithms and apply statistical tests to determine the likelihood that different methods return the same results. Derrac et al. [85] present a survey of statistical methods for comparing the performance of evolutionary algorithms. Garcia et al. [86] provide a similar study of non-parametric tests for comparing results across a variety of datasets. Ali et al. [87] show a variety of techniques for visualizing results of stochastic algorithms.

For comparisons, we use a Mann-Whitney U test [88], a non-parametric method for comparing two independent sample groups without requiring assumptions of samples being normally distributed (as a t-test does). However, despite the lack of assumption of normality, it has only 5% efficiency loss (efficiency here is the quality of the estimator) compared to the t-test on normally distributed data, and far more efficient than the t-test when we draw the data from a non-normal distribution. The test determines the likelihood that one group of samples of a random variable A is stochastically less than another group of samples of a random variable B:

$$Pr(A > x) \leq Pr(B > x) \; \forall x \in (-\infty, \infty). \tag{5.1}$$

5.2.3 How to Define a Successful Crossover

Uy et. al. [89] define a constructive crossover between two parents as one where both offspring produce fitnesses better than their parents. On a variety of benchmark problems, they find these rates to be between 4% and 12%. In a multi-objective environment, this is a bit harder to define, so we invert the problem and define what an unsuccessful crossover is.

In a multi-objective search, we use Pareto-optimality to rank our individuals in the

population as a measure of fitness. In this context, we define an unsuccessful crossover between two parents is one where at least one parent strictly dominates each child. We define a successful crossover as one that is not unsuccessful, meaning at least one child is non-dominated by the parents. We refer to this definition of success and failure as parent-based. Code for implementing this definition of an unsuccessful crossover is shown in Code Listing 5.1, while the implementation for a successful case is shown in Code Listing 5.2. We use these implementations to perform an evolutionary simulation in Section 5.2.5.

Code Listing 5.1: Implementation of an Unsuccessful Mating Parent-Based

```
def unsuccessful_crossover(ind1, ind2):
    """
    Produces two children that are each dominated
    by at least one parent
    """
    worse_children = []
    while len(worse_children) < 2:
        x = ind1.mate(ind2)
        if dominates(ind1, x) or dominates(ind2, x):
            worse_children.append(x)
    return worse_children
```

Code Listing 5.2: Implementation of a Successful Mating Parent-Based

```
def successful_crossover(ind1, ind2):
    """
    Not unsuccessful_crossover, meaning
    at least one child is at least codominant
    """
    not_worse_children = []
    # In this case we need to generate just one
    # individual that violates our constraint
    while len(not_worse_children) < 1:
```

```
        x = ind1.mate(ind2)
        if not (dominates(ind1,x) or dominates(ind2,x)):
            not_worse_children.append(x)
    # Once we have our one good child,
    # we can get any other random child
    not_worse_children.append(ind1.mate(ind2))
    return not_worse_children
```

We can compare this to another method of classifying success of crossover that is population-based. Rather than looking only at parents and children, we can also examine the rest of the population at the time of crossover. In this context, in a minimization scheme, a failed crossover is one in which both children appear at a higher (worse) Pareto rank than that of the better parent (or both if the parents are of the same rank). A successful crossover is one in which the parents produce at least one child at the same Pareto rank as, or lower than, the better parent. While this definition is more computationally expensive to simulate (i.e., it requires computing dominance interactions across multiple individuals rather than just the parents), it captures the time-varying and population-relative aspect of the evolutionary process. Example code for the implementation of an unsuccessful crossover for this population-based criteria is shown in Code Listing 5.3, and the implementation for successful crossover is shown in Code Listing 5.4. Both of these code segments assume that ranks are computed for the population before we use these functions.

Code Listing 5.3: Implementation of an Unsuccessful Mating Population-Based

```
def unsuccessful_crossover_pop(ind1, ind2, population):
    """
    Produces two children that are of worse pareto rank
    than parents
    """
    # First let's find the better rank of the parents
    target_rank = min(ind1.rank, ind2.rank)
```

```
    # Now let's extract the front fitness array
    # from the population
    target_front_fitness = np.array([
       ind.fitness for ind in population \
       if ind.rank == target_rank])
    worse_children = []
    while len(worse_children) < 2:
        x = ind1.mate(ind2)
        # Compute a dominance array, this will tell if
        # any individual in the front dominates the child
        dominance = \
          np.all(target_front_fitness<=x.fitness,axis=1)\
          &np.any(target_front_fitness<x.fitness,axis=1)
        # If any individual in this front dominates the
        # child, it is unproductive
        if np.any(dominance):
            worse_children.append(x)
    return worse_children
```

Code Listing 5.4: Implementation of a Successful Mating Population-Based

```
def successful_crossover_pop(ind1,ind2,population):
   """
   Not unsuccessful_crossover, meaning at least one
   child is at the same rank or better with the parents
   """
   # First let's find the better rank of the parents
   target_rank = min(ind1.rank, ind2.rank)
   # Now let's extract the front fitness
   # array from the population
   target_front_fitness = np.array([
     ind.fitness for ind in population \
     if ind.rank == target_rank])
```

```
    not_worse_children = []
    # In this case we need to generate just one
    # individual that violates our constraint
    while len(not_worse_children) < 1:
        x = ind1.mate(ind2)
        # Compute a dominance array, this will tell if
        # any individual in the front dominates the child
        dominance = \
          np.all(target_front_fitness<=x.fitness,axis=1)\
          &np.any(target_front_fitness<x.fitness,axis=1)
        # If no individual dominates this child,
        # it is productive
        if not np.any(dominance):
            not_worse_children.append(x)
    # Once we have our one good child,
    # we can get any other random child
    not_worse_children.append(ind1.mate(ind2))

    return not_worse_children
```

5.2.4 Performance of the Optimization

For a two-objective problem, we can measure the performance of the evolutionary process by tracking the area under the curve (AUC) of the non-dominated frontier over the generations of the optimization. This methodology extends into higher dimensions by computing hypervolumes under a non-dominated hypersurface. Equation 4.1 showed this definition in a two-dimension minimization problem.

In the case of the population-based definition of failure and success, AUC only decreases in one of two ways as the evolution proceeds:

1. There is a successful crossover, and at least one of the parents was a Pareto individual. In this case, there is always an AUC improvement.

2. There is a successful crossover with two non-Pareto parents, but the random child they produce happens to be a Pareto individual.

To track the stochastic nature of the evolutionary process, we can repeat the optimization over many trials, each trial yielding a different AUC curve over time. We can then compare these populations to each other. In the case of this population-based definition, we can expect the average AUC across these trials to perform as follows:

At generation i, let A_i be the AUC and P_i be the number of parental pairs containing a Pareto individual. Assume we have an initial AUC A_0 and initial count of P_0 parental pairs containing at least one Pareto individual.

After generation 1, our AUC can be given an expected upper bound (in minimization) by subtracting out expected improvements:

$$A_1 = A_0 - P_0 Pr(\text{success})\alpha A_0 = A_0(1 - P_0 Pr(\text{success})\alpha),$$

where we let αA_i be expected AUC contribution at generation i when there is a decrease in AUC. Because we assume our standard deviation shrinks as our objectives shrink, the AUC contribution is not a constant; each improvement contributes a constant proportion of the AUC.

Continuing this process,

$$A_i = A_{i-1}(1 - P_{i-1} Pr(\text{success})\alpha),$$

so,

$$A_2 = A_1(1 - P_1 Pr(\text{success})\alpha) = A_0(1 - P_0 Pr(\text{success})\alpha)(1 - P_1 Pr(\text{success})\alpha).$$

We can, therefore, rewrite A_i as

$$A_i = A_0 \prod_{k=0}^{i-1} (1 - P_k Pr(\text{success})\alpha). \tag{5.2}$$

Hence, A_i results in an exponential decay in AUC as generations increase. We also expect that probability of success is proportional to the rate of decay, and thus a linear change in the probability of success has an exponential impact on AUC.

The parent-based criteria should behave similarly; however, a crossover success on a Pareto individual does not guarantee an AUC improvement, because another existing individual may dominate the resulting child in the population. We can think of this as another probability, $Pr(\text{non-dominated})$ being multiplied by the $P_k Pr(\text{success})\alpha$, which will result in a slower decay.

5.2.5 Evolutionary Simulation

Because we are after a relationship between crossover success rates and AUC convergence, we choose not to implement an underlying genome. Instead, we rely solely on the phenotype (expression) of that genome in objective space. Our definitions of success and failure from the previous section are based only on objective scores; therefore, we can set rates of success and failure and draw samples from assumed distributions to simulate the crossover process. This abstract process should yield more general results than relying on a specific genome, crossover operator, and evaluation function.

In this simulation, we make several assumptions:

1. Objective scores are continuous.

2. The objective scores of a child are normally distributed around the centroid of the objective scores of its parents.

3. To simulate the convergence of the evolutionary process, we use a standard deviation proportional to the mean of the normal distribution.

4. Objective scores are bounded on $[0, 1]$, effectively creating a truncated normal through rejection sampling of those objectives that lie outside of these bounds.

5. Each pair of parents produce exactly two children during the crossover operation.

Because we do not have an evaluation function, our evolutionary simulation only consists of selection and mating. For selection, we chose the non-dominated sorting genetic algorithm (NSGA), which involves sorting the combined N parents and N children by Pareto ranking and selecting the top N individuals to be the parents for the next generation, where N is the population size. We chose this method for its inherent elitism benefits, i.e., as long as there are less than N Pareto individuals, the Pareto frontier is preserved from generation to generation.

We perform the crossover operator as follows. For each pair of individuals (randomly chosen without replacement), a random number is generated uniformly on [0,1]. If the random number is less than the given probability of failure p_i, then the prescription for an unsuccessful crossover is used (either Code Listing 5.1 or Code Listing 5.3). Likewise, if the random number is greater than p_i, then a successful crossover is performed (Code Listing 5.2 or Code Listing 5.4).

We do not use any mutation in our evolutionary simulation due to the lack of an underlying genome. Instead, we assume the statistical nature of our crossover operator captures the variations a mutation operator would offer.

Code Listing 5.6 shows our definitions for an individual and its mating operator.

5.2.6 The Experiment

We set up a two-objective problem, with our objectives as being independently distributed from an initial normal distribution with a mean of 0.5 and standard deviation of 0.1.

We construct a mating operator that produces children with objective scores from a normal distribution centered at the mean of the parents and a standard deviation equal to

one-fifth of the mean. We simulate a truncated normal, since our objectives are bounded on [0,1] by regenerating any solutions that fall outside that range.

We use a population size of 128 individuals, each with their objectives drawn from the initial distribution. The evolution runs for 100 generations and is repeated 50 times for each probability of unsuccessful crossover from (30%-80%). For each probability of unsuccessful crossover p, the probability of a successful crossover is $1 - p$.

Code Listing 5.5: Fitness Generation Through Rejection Sampling

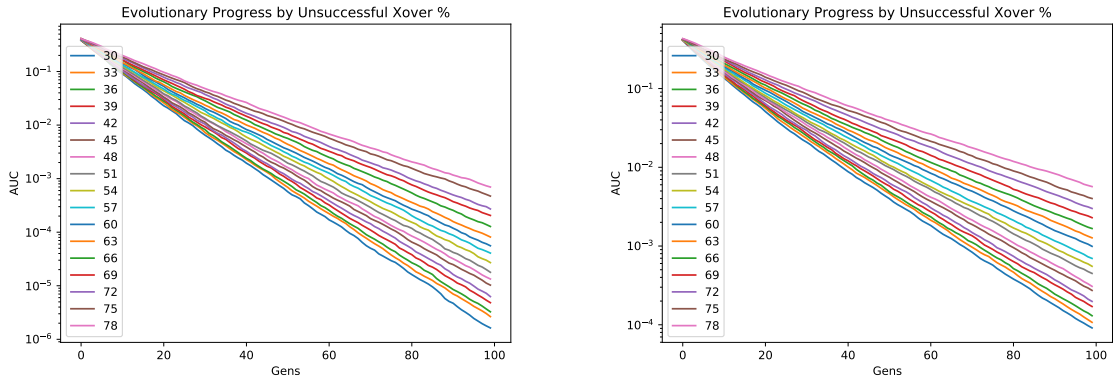
```
def gen_fitness(mu, sigma):
    continuous = []
    # Begin rejection sampling
    while len(continuous) < 2:
        i = len(continuous)
        # Generate objective score sample i
        test_var = sigma[i]*np.random.randn(1) + mu[i]
        # Test to see if within objective bounds
        if test_var[0] < 1 and test_var[0] > 0:
            continuous.append(test_var[0])
    continuous = np.array(continuous)
    return continuous
```

Code Listing 5.6: Class definition of an Individual

```
class Individual:
    def __init__(self, mu=np.array([0.5, 0.5])):
        sigma = mu/5.
        self.fitness = gen_fitness(mu, sigma)
        self.rank = None

    def mate(self, other):
        avg = (self.fitness+other.fitness)/2
        return Individual(mu=avg)
```

5.2.7 Results



(a) Population-based success criteria

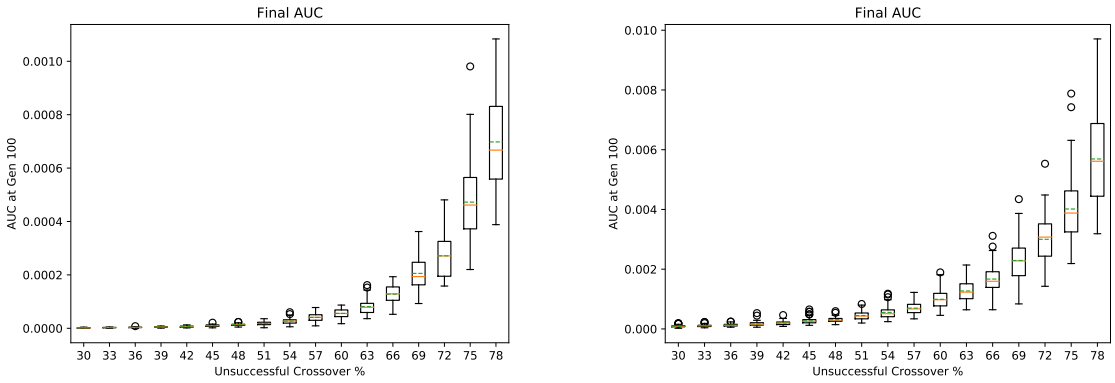
(b) Parent-based success criteria

Figure 5.8: Log plots of the area under the non-dominated frontier at each generation averaged over 50 trials, where each line is a difference in unsuccessful crossover percentage.

Figure 5.8 shows the AUC of the non-dominated frontier averaged over 50 trials for each probability of failure. We only plot every 3% for clarity. We also chose a log-plot to show that each 50-trial experiment has a decaying exponential response over the course of the 100 generations, and the 3% differences in unsuccessful crossover probability are roughly linearly spaced, suggesting this probability has an exponential effect on convergence for both success criteria. The stricter population-based criteria creates a faster response than the more relaxed parent-based criteria, which is more clearly illustrated in Figures 5.9 and 5.10.

Figure 5.9 uses box and whiskers plots to capture the full range of the stochastic process. We can see the exponential characteristic of the median (solid orange line) of each probability of failure. The population-based success criterion decays more quickly than the parent-based criteria as the rate of unsuccessful crossover decreases. Figure 5.10 illustrates that the slopes of the log plots change with the probability of crossover failure. The stricter criteria enforce a faster convergence. We can also clearly see the exponential relationship between probability of success and failure and convergence of the optimization.

Figure 5.11 compares the ability of the population-based and parent-based definitions



(a) Population-based success criteria

(b) Parent-based success criteria

Figure 5.9: Box and whisker plots of the final AUC at generation 100 by crossover failure percentage. Each box and whisker represents the results of 50 trials. The effect of crossover success is an exponential response on AUC. Orange solid lines show medians, green dashed lines show means.

of success to improve AUC over a generation in terms of probability of failure. We compute the probability of improvement by loading all trials for a given experiment and counting the number of times the AUC changes from one generation to the next, out of all generations and all trials for the probability of failure.

Figure 5.12 shows the number of generations required for a probability of failure to have a statistically significant impact on AUC for a given number of trials. We simulate each probability of failure for the full 100 generations and 50 trials; we call each of these an experiment at a given failure level. Of the 50 trials per experiment, 15, 20, and 25 are chosen randomly to observe the impact that the number of trials has on the statistical significance. Next, we iterate through each potential separation of the probability of failure from 1% to 20%. This separation represents the percent improvement one crossover has over another.

For each possible separation, we use a Mann-Whitney U test for each of the 100 generations to compare the k trials of one probability to the k trials of the other. The 100 comparisons of k trials create a p-value curve for each possible delta that we then average. For example, if our delta is 2, we start creating p-value curves for 80 vs. 78, then 79 vs. 77,

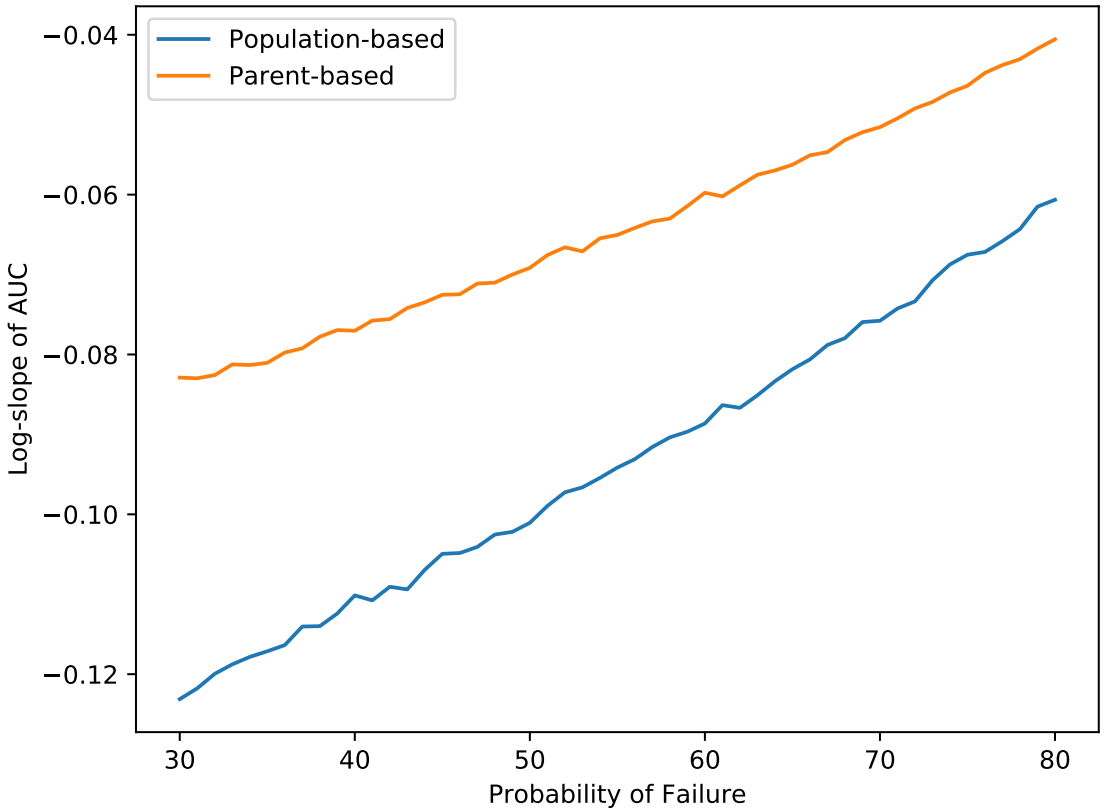


Figure 5.10: Slope of AUC log plot vs. the probability of failure of crossover. The population-based criteria show a faster rate of decay of the AUC, and responds more quickly to changes in probability of failure.

through to 32 vs. 30. This yields 49, 100-point p-value curves for our chosen separation. We ran 51 full experiments, with each separation δ yielding $51 - \delta$ curves to average. We now average all the curves for a given separation and number of trials. Figure 5.13 shows an example of these average p-value curves for a static 30 trials per comparison and separations of crossover success rates between 1 and 10. The final step is to compute the final generation at which the average p-value falls below and stays below the critical threshold, set at 0.025, representing a one-tailed 97.5% confidence rate that one distribution has a lower mean than the other. The generation that the last crossover occurs at is the number of generations that a given separation needs to have a statistically significant impact (at the 97.5% level).

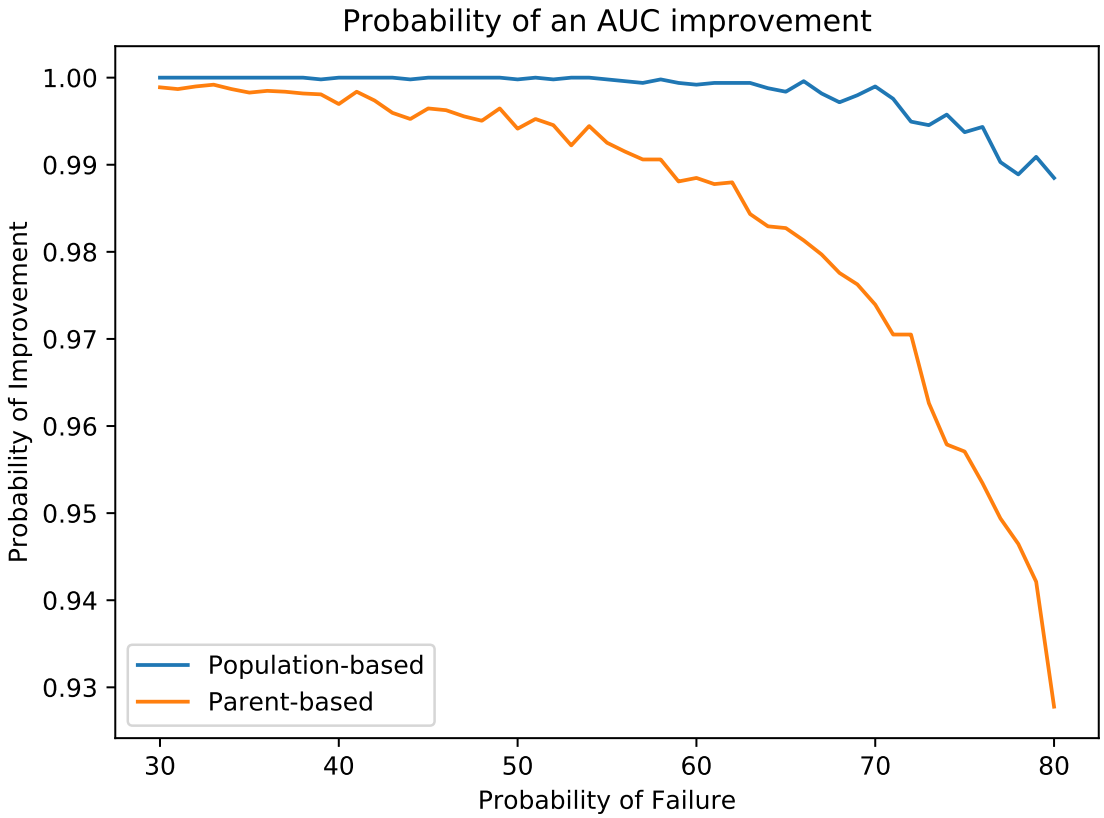
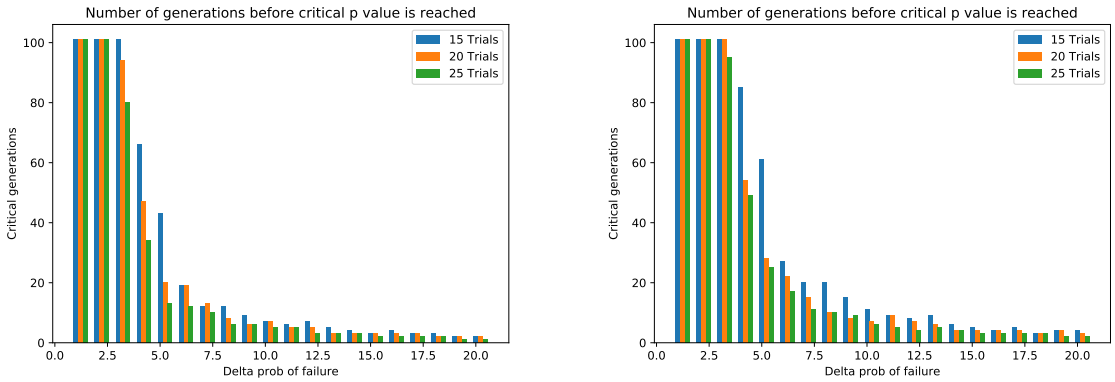


Figure 5.11: Probability of an AUC improvement from one generation to the next.

Figure 5.14 compares the impact of the two different success criteria side-by-side. These plots are meant to be decision aids for designing experiments, i.e., for a known amount of crossover improvement and generations of the evolutionary process, these plots show a minimum number of trials required to have a statistically significant impact on AUC. While the two criteria behave similarly, the stricter population-based success criteria have a significant impact sooner, showing improvement in AUC with 2% difference in crossover success and approximately 40 generations. On the other hand, the parent-based success criteria requires a 3% difference in crossover success and about 30 generations to show improvement. Population-based crossover also sees the number of required trials fall off more quickly than the parent-based criteria.

The plots in Figure 5.14 were generated similarly to those of Figure 5.12, but rather than use a fixed number of trials and generations, we begin with the full 50 trials for each δ and



(a) Population-based success criteria

(b) Parent-based success criteria

Figure 5.12: Comparison of how many generations a given change in probability of success will take to have a statistically significant impact on AUC, for three different numbers of trials.

number of generations. We compute the average p-value across all experiments separated by δ, reducing the number of trials until the p-value is no longer below the critical value of 0.025. The last number of trials before this happens is the minimum number of trials required to generate a statistically significant separation of AUC for a given number of generations and crossover improvement.

5.2.8 Conclusion and Future Work

From our results, it is clear that the statistical power of a crossover operator on AUC improvement depends on the number of trials, generations per trial, and the amount of crossover improvement. The impact of choosing NSGA (or any mechanism with elitism) for selection in the case of the population-based definition of success is that the inherent elitism strongly affects the amount of AUC improvement in each generation, due to the independent probabilities of selecting a non-dominated individual and generating a successful child. Anytime both these events occur, there is an AUC improvement. Therefore, as the non-dominated front grows, so will the amount of AUC improvement in each generation.

In Appendix A, we design a "matchmaker" operator to improve crossover. The objec-

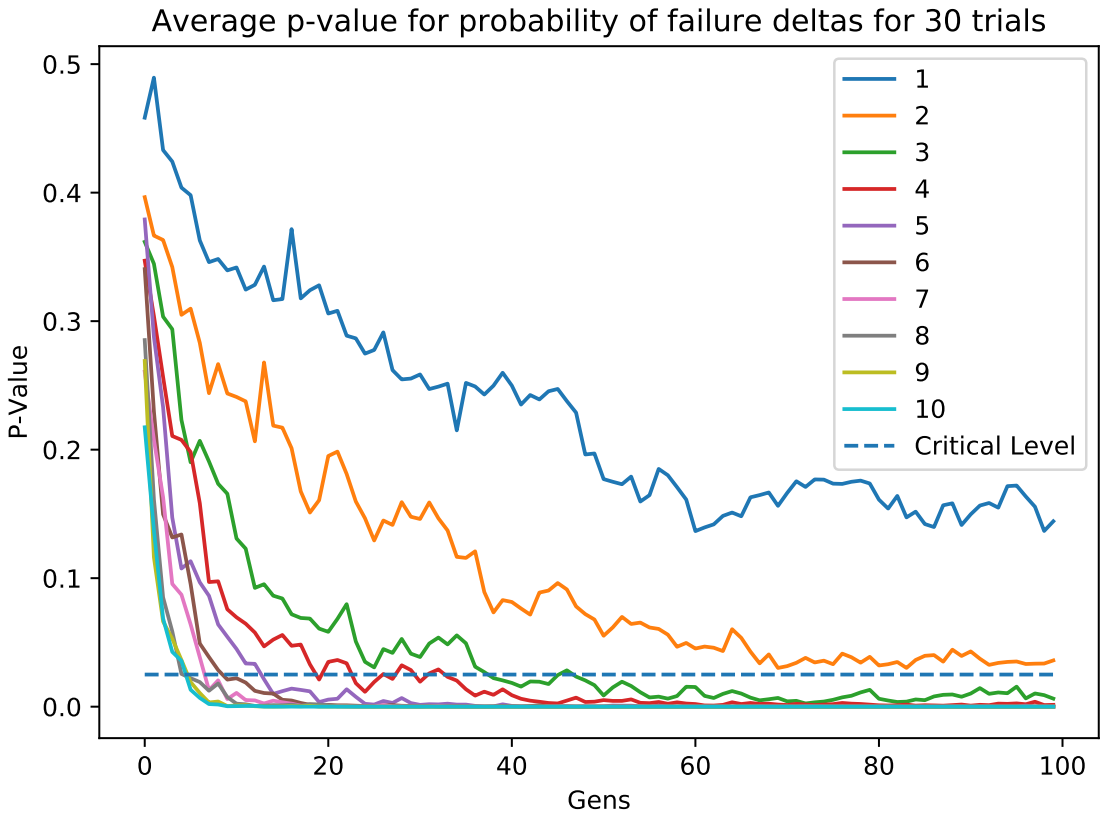
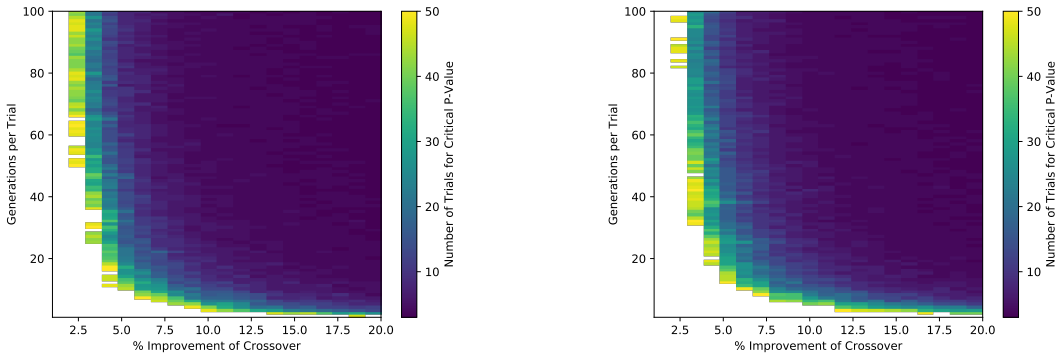


Figure 5.13: Average p-value for the Mann-Whitney U test at each separation of failure probability for a constant 30 trials, where each line indicates a difference in probability of failure rate. This particular plot is done using the data from the population-based success criteria.

tive is to replace the random shuffling of parents into pairs with a learned strategic pairing methodology. The results in the previous section inform our design of experiments.

As an example, take Figure 5.14a, which shows results with the population-based success criteria. If a matchmaker can increase the probability of a successful crossover by 3%, and we would like to run no more than 30 trials of our evolution, Figure 5.14a shows we would need to run each trial for 44 generations before we would observe a statistical improvement in AUC. On the other hand, if the matchmaker can achieve as much as a 6% improvement in success, we can see statistical improvement of the evolutionary process measured by AUC in only 13 generations in the same number of trials. For the parent-based criteria, these we would require 66 and 14 generations respectively, using the same



(a) Population-based success criteria

(b) Parent-based success criteria

Figure 5.14: Design of experiment plots showing the number of trials required for critical p-value to show significant difference in AUC for a given number of generations and % improvement of crossover.

two probability improvements and the number of trials.

For some problems, especially those that are more easily solved by an evolutionary algorithm, knowledge of this dependence on generations is helpful. For instance, if training this matchmaker takes data from a certain number of generations before it performs accurately, the number of generations to show improvement represents the number of generations at which we observe a reward for using a matchmaker. If the percent improvement is low enough, or the problem is easy enough that the evolutionary algorithm can solve it, the matchmaker may never be able to have an impact on the process.

In addition to using the results of this simulation as an informative tool, there is potential future work in improving the simulation itself, including:

- Experimenting with other statistical properties when executing a crossover, such as correlated objectives or varying distributions and parameters.
- Increasing the resolution that the simulation is run over (better than 1% step size in improvement).
- Implementing other definitions of success and failure, such as a child contributing to AUC improvement as a metric.

5.3 Summary

This chapter aimed to improve the selection and mating steps of the evolutionary process. The algorithms for these steps have historically favored fast performance due to the evolutionary operators taking a significant amount of the processing time during the optimization. Due to EMADE 's high-level primitives increasing evaluation time, we investigated these new methods in hopes that we could create better performing operators in terms of number of successful children produced at the cost of some idle time in the evolutionary loop.

The selection method we investigated used a novel concept of probabilistic dominance based on individual test-case performance. This was moderately successful, although that success appeared to depend on balance of distributions of the gene pool for its objectives. Future work could follow this thread and perhaps alter some of the assumptions, such as the truncated normal distribution we applied through Markov-chain Monte Carlo.

CHAPTER 6
CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

We introduced EMADE as a powerful autoML tool. EMADE differentiates itself from the rest of this emerging field through its use of bio-inspired processes to combine proven techniques from the areas of machine learning and signal processing. In this thesis we demonstrated how these bio-inspired processes are able to evolve human-competitive solutions on challenging problem sets. EMADE supports the generation of algorithms for imagery and time-domain problems, a capability not found in the rest of the autoML space.

A concept of tiered datasets allowed us to process computationally expensive datasets without sacrificing the ability to evaluate large numbers of individuals. The tiered dataset approach allowed autoML to cross into new domains, such as imagery with the xView dataset or signal processing with bathymetric LIDAR simulations.

We demonstrated success across a suite of problems covering feature data and stream data for both classification and regression. The key to solving these problems is to continuously grow the capabilities of EMADE by adding new primitives from the literature and better understanding what drives the evolutionary process.

Finally, with a paradigm shift of long evaluation times, we explored new ways to exploit our wait time for the main evolutionary loop to increase our probability of success from matings and mutations. In particular, the fuzzy selection with probabilistic dominance showed promise.

One of the most powerful examples on how the contributions in this dissertation come together is in the bathymetric LIDAR application from Section 4.4. For this problem, we started with a high-fidelity simulation and industry standard algorithm for processing

LIDAR returns, and evolved a significantly better solution. What is most illuminating about the solution discovered by EMADE is that it was not only simpler, but it used techniques from an entirely different domain, i.e. image processing. This novelty speaks to EMADE's ability to grow over time as more primitives and data types are added to the code base from a variety of domains, as well as its ability to dispassionately pursue non-traditional ideas that might not even occur to researchers in narrow fields.

6.2 Future Work

As of Fall 2008, Georgia Tech supports a Vertically Integrated Project course in automated algorithm design, almost fifty students actively applying and modifying the EMADE code base to solve new problems that interest them and improve understanding of the tool.

We wish to continue down important areas of research with EMADE, including implementation of many objective techniques (i.e. much greater than three), exploring speciation and novelty techniques, and using EMADE as a platform for other thoughts on evolutionary computation.

One particular area in much need of attention is the field of deep learning. Due to high computation expense, deep neural network support in EMADE is currently limited. We would like to further explore this area and find clever ways to incorporate this powerful machine learning tool into EMADE's space of primitives, and apply evolutionary concepts to the black art of neural network architecture as we have applied these evolutionary concepts to the greater space of machine learning.

Other potential areas for improvement include bloat removal, speed improvements through identification of bottlenecks in EMADE, and leveraging hardware such as GPUs, as well as other concepts to speed up the evaluation times. One concept that could considerably speed up evaluation times in EMADE is the implementation of a co-evolutionary process to select test cases from the dataset for computing objective scores against, rather than use the full dataset for the entire optimization. The co-evolution would allow the sec-

ond evolutionary process to find the smallest set of test cases that best approximate the performance of the full dataset. Another interesting thread of research could be the exploration of how to best use the information gained from analyzing statistics on objective scores across the cross-folded datasets in EMADE. For example, rather than only using averages, we could integrate the use of variances across aggregate scores of an individual to get a better measure of the tendency of an algorithm to overtrain. Identifying algorithms that have large swings in objective scores across folds could also help terminate evaluation early, instead of evaluating across all folds.

Appendices

APPENDIX A
MATCH MAKING

All evolutionary algorithms rely on the concept that making small changes to successful individuals yields better individuals over time. A principal operator in an evolutionary algorithm is a crossover or mating operator that can exchange information between individuals to construct new ones. A problem in which crossover aids convergence is particularly well-suited to the evolutionary process because selecting successful individuals and exchanging information yields better individuals. However, most evolutionary algorithms randomly pair individuals in the population after selecting a set of parents. There must be a more intelligent manner to pair individuals to increase the probability of producing successful offspring. For example, humankind has managed to produce all varieties of domesticated dog breeds in the last 14,000 years through selective breeding, a relatively brief amount of time on the evolutionary timeline. Section 5.2 showed that if we can improve the probability that at least one child is non-dominated by its parents, we can outperform the traditional random pairings of parents.

This section explores the concept of matchmaking to strategically pair parents. We introduce a machine learner into the evolutionary process after selection. The machine learner is responsible for the pairing of parents before crossover.

A.1 Features of Crossover

Ideally, we design the matchmaker to be a black box that predicts whether or not a crossover will be successful from a number of features. To best capture crossover dynamics, we select features that cover both phenotypic and genotypic similarities and differences of the parents. We designed all features to be symmetric, meaning crossover(parent1, parent2) has the same set of features as crossover(parent2, parent1). We looked at a two-objective

problem (which defines a subset of phenotypic features that we may exploit). The genotypic features we use are:

- Height difference: The absolute value of the difference in heights of the trees representing each parent.
- Similarity: A measure of similarity of the two parents. This score counts the number of elements the parents have in common, starting at the root of the tree of each parent.
- Difference in number of elements: The absolute value of the difference between the number of elements in both of the parents.
- Primitive similarity: A histogram is constructed for each parent with one bin for each primitive in the primitive set. Each histogram represents how many times each primitive appears in the tree of the parent. The feature is the L^2 norm of the difference between the histograms of the parents.
- Total primitives: With histograms constructed in the same manner as for primitive similarity, "total primitives" is computed as the L^2 norm of the sum of the histograms of the parents.
- Edit distance: This is the most expensive feature we implemented. It represents the number of changes needed to change one parent into another. This was computed by the zss python package, which implements the Zhang and Shasha tree edit distance [90].
- Vectorized histogram differences: Here we construct N features, where N is the number of primitives in the primitive set. This set of features is the absolute value of the difference between each parent's count of each primitive.

The Phenotypic features we use are:

- Fitness difference: The L^2 norm of the difference between vectors of objective scores for the two parents.
- Fitness difference X: The squared difference of the first objective score.
- Fitness difference Y: The squared difference of the second objective score.
- Test case covariance: For the next several features, let u be the vectorized test case predictions of parent 1 and v be the vectorized test case predictions of parent 2. The test case covariance is the correlation coefficient of the vectorized test case predictions of the two parents.
- Test case difference: The L^2 norm of the difference of the vectorized test case predictions of the two parents, $||u - v||_2$.
- Cosine difference: The cosine distance measure computed from the vectorized test case predictions of the two parents:

$$1 - \frac{u \cdot v}{||u||_2 ||v||_2}.$$

- Barycurtis difference:

$$\frac{\sum |u_i - v_i|}{\sum |u_i + v_i|}.$$

- Canberra difference:

$$\sum_i \frac{|u_i - v_i|}{|u_i| + |v_i|}.$$

- Chebyshev difference:

$$\max_i |u_i - v_i|.$$

- Cityblock difference: Also known as the Manhattan distance

$$\sum_i |u_i - v_i|.$$

- Minkowski difference:

$$||u - v||_p = (\sum |u_i - v_i|^p)^{1/p}. \left(\sum w_i(|(u_i - v_i)|^p)\right)^{1/p}.$$

- Square Euclidean difference:

$$||u - v||_2^2.$$

- Sum of distance to origin: The L^2 norm of the sum of the vectors of objective scores for the two parents.

- Age difference: The difference in the generations in which each parent first appeared.

A.2 *In Situ* Learning

To produce a matchmaking algorithm, we experimented with a number of different machine learners to identify which would be appropriate. The next subsection explores the use of an XGBoost machine learning algorithm configured to perform a logistic binary classification. Section A.4 explores using EMADE to optimize the algorithm for matchmaking.

For *in situ* learning that the next section explores with various test problems, we use a machine learner as follows. First, each evolutionary trial begins without matched crossover for N_{prime} generations. During these generations, individuals crossover randomly, as they normally would. As each crossover occurs, the features from Section A.1 and outcome of the crossover (success for at least one child that is not dominated by both parents) are logged in a database. After N_p generations, we train the classifier for the first time on the features produced during those generations. The matchmaker updates every N_m generations, and optimizes with a grid search every N_g generations. The grid search uses the parameters n_estimators , varying across $\{10, 20, 50, 100, 150, 170, 180\}$, learning_rate , varying across $\{0.01, 0.1, 0.3, 0.5, 0.8, 1.0\}$, and max_depth, varying across $\{3, 6, 9\}$. This parameter space represents 126 possible XGBoost classifiers, from which we select the

best configuration.

After the learner is trained for the first time, we use the learner after selection and prior to mating to order the population. We create an $M \times M$ matrix C, where M is the size of the population, to represent the probabilities of all potential crossovers. We compute the features of the current population and feed them to the machmaker to predict a probability of success for each possible crossover in the population. We set C_{ij} to be the probability that the crossover of individuals i and j produces a child that is non-dominated with i and j, which the matchmaker predicts. Next a greedy approach pairs the population. We sum across each row in C to create a vector measuring the utility of each individual. The individual corresponding to the highest value in this vector is expected to have the highest overall probability of producing a non-dominated child, and the lowest value should correspond to the individual we least expect to produce a non-dominated child. We sort the individuals in descending order by these values. Next, we intitialize a list of elligible mates as the entire population, and a list of matched individuals as empty. Starting with the first individual in the sorted list, we adjust the corresponding row of C by a fixed baseline probability p_b. We then normalize the row by dividing by its sum. We use this resulting vector to perform a weighted random choice of an individual with which to pair. We remove both individuals from the elligible mates list and add them to the list of matched individuals. We repeat this process until the elligible mates list is empty. Each time we extract the row from C, we choose only the columns that correspond to elligible mates prior to performing the normalization and random choice.

This greedy matching procedure should give the parents most likely to produce non-dominated offspring the first preference of mates, while still incorporating some of the randomness that evolution requires. The next subsection shows that despite this matching procedure, performance of the evolutionary procedure does not improve. Section A.4 investigates some of the reasons this procedure does not succeed.

Table A.1: Primitive Set for the Word Count Problem

Logical	**Arithmetic**	**String**	**List**
and	add	split	length
or	subtract	strip	concatenate
less than	multiply	replace	
greater than		concatenate	
not		upper	
less than or equal		lower	
greater than or equal		capitalize	
equal		center	
not equal		count	
exclusive or			

A.3 Test Problems

We explored several test problems with our matchmaking framework. These problems were: the parity problem, an eight bit comparator problem, PNP uniform length and uniform frequency [91], and a word count problem. In each problem, the matchmaking algorithm failed to improve performance over the random pairing of individuals. This section walks through the word count problem to understand why the performance suffers.

For the word count problem, we used the full text of *Romeo and Juliet* from Project Gutenberg. Each line of text becomes a test case, where the number of words on that line serve as the truth data. Table A.1 shows the functions that constitute the primitive set. The terminal set comprises each character in the alphabet (upper and lowercase), the digits 0-9, quotes, apostrophes, backslash, comma, period, newline, pound sign, true, false, and an empty set. To avoid the optimization solving the problem too quickly, we exclude the space character from the terminal set.

To score the word count problem, we use two objectives: mean absolute error and variance of error. Error is measured as the difference between the predicted and actual word count of each line. For selection, we use a traditional NSGA-II procedure. We chose a population size of 128 individuals, a probability of crossover of 90%, and a probability

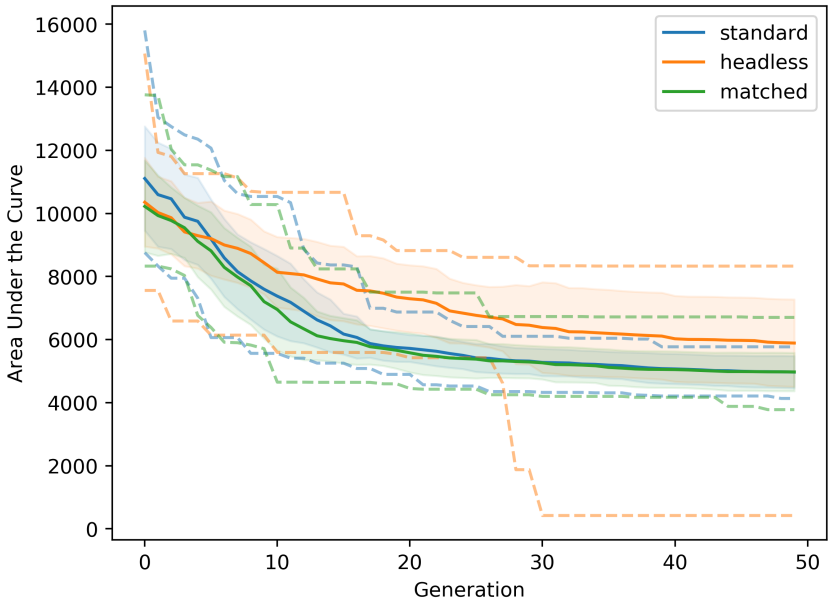


Figure A.1: Average AUC curves for standard, headless, and matched crossover over 30 trials. The shaded area represents $\pm\sigma$, the dashed lines represent the best and worst case AUC at each generation.

of mutation of 10%. We run 30 trials of 50 generations each for the cases of standard crossover, headless crossover, and matched crossover. The headless crossover is where all crossovers occur with randomly created trees; we include this to demonstrate the benefit of crossover. For the matched crossover trials, the *in situ* matchmaker used the parameters of $N_p = 5$, $N_m = 5$, $N_g = 10$, and $p_b = 0.01$.

Figure A.1 shows the performance of the three methods of crossover. Each line traces the average AUC over 30 trials that the non-dominated individuals create on the objectives of mean absolute error and variance. The shaded area around each line represents plus or minus one standard deviation, and the dashed lines represent the best and worse AUC for each generation. Both standard and matched crossover outperform headless crossover, which means the evolutionary process benefits from the mating operator. Theoretically, we require crossover to be beneficial for a matchmaker to function properly. However, we also see that the standard and matched crossover have nearly identical AUC curves.

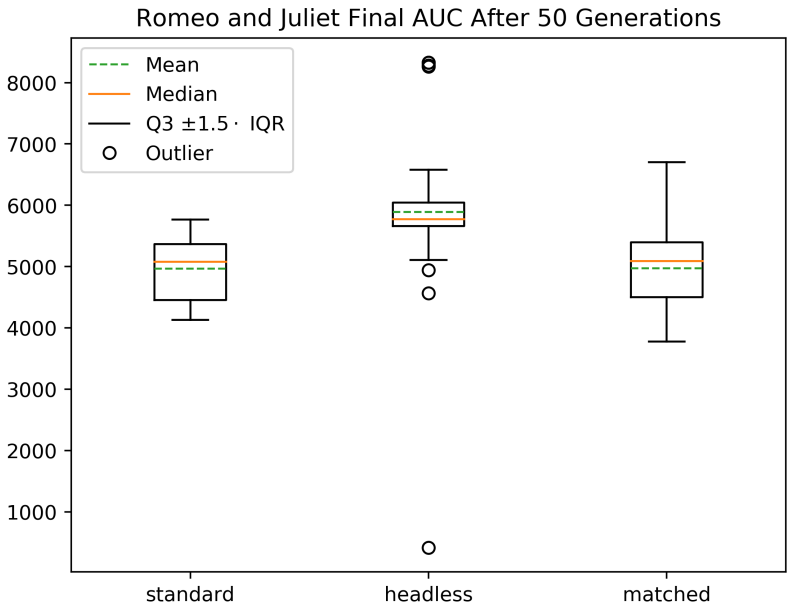


Figure A.2: Box and whisker plot of distribution of AUC at generation 50 over 30 trials.

Figure A.2 contains box and whisker plots for standard, headless, and matched crossover performance respectively at the end of the 50th generation. Again, both standard and matched crossover outperform headless crossover. Although the interquartile range between standard and matched are similar, the whiskers of matched crossover extend farther in each direction, indicating a more variable process. In general, we prefer tighter performace characteristics, which indicates more reliability.

Figure A.3 shows the probability at each generation that matched and standard, as well as headless and standard, have different distributions. We compute these p-values with the same Welch's t-test used in Section 5.2. We can reject the null hypothesis that headless and standard come from the same distribution at more than the 99% confidence level. On the other hand, we cannot reject the null hypothesis that standard and matched crossover yield different distributions.

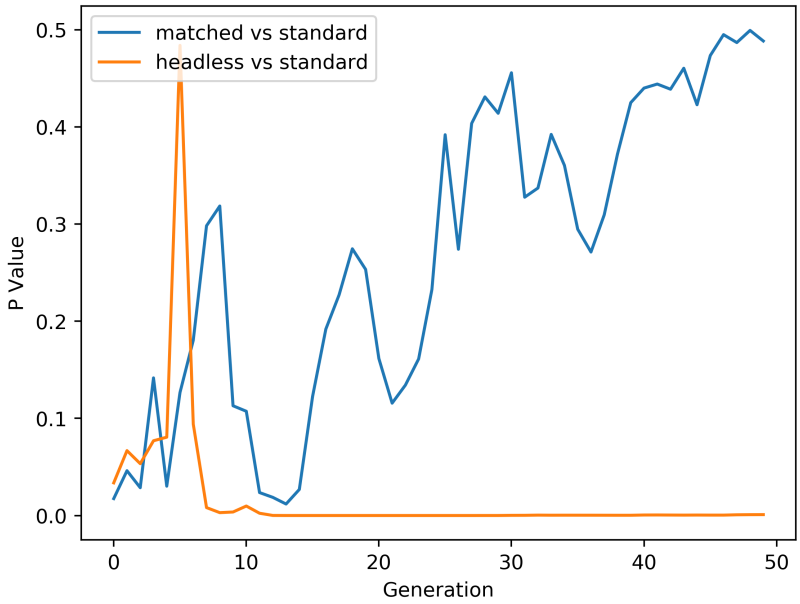


Figure A.3: P-Values from a Welch's t-test to reject the hypothesis that the distributions of AUC at each generation are equal.

A.4 Evolving a Matchmaker

This section explores whether the classifier is the source of the difficulties found in the previous section. We allowed the classifier to train on accuracy in the *in situ* method, which favors the surpression of false positives, because crossover is more often unsuccessful than successful. If false positives are surpressed, the classifier is more likely to predict false negatives. A false negative during the evolutionary process could be far more detrimental to the performance of the search because it represents an unexplored solution. Given that we observed that the parent-based success rates hovered between 10 and 20 percent, we hypothesized that we could create a classifier with EMADE, trained on a large number of crossovers generated from different trials, which has a better accuracy than the naïve random pairing approach (which has a low accuracy, high false positive rate, and false negative rate of 0), while still maintaining a low false negative rate.

We tested our theory by tracking these features through 60 trials of an evolutionary ex-

periment of 50 generations each. The population size was set to 128. With a 90% crossover rate, we expect to generate $E[x] = 0.9 \cdot \frac{128}{2} \cdot 50 \cdot 60 = 172,800$ crossovers with which we can study and build predictive models. Because our goal is to develop a binary classifier for success and failure of crossover, we supervised this dataset with the parent-based definitions of success and failure shown in Code Listings 5.2 and 5.1, respectively. We chose this because both parent-based and population-based definitions showed improvement in a similar number of trials, but from a machine learning perspective, parent-based definitions result in more common successes. While the dataset is still unbalanced in favor of failures, the successes result in significantly more observable positive instances, and allows a classifier to have a larger false positive rate at the same positive predictive value.

We produced two datasets from eight trials: a set for developing matchmaking algorithms in EMADE, and a set for validating the algorithms EMADE produced. The training set consisted of two trials, and the validation set consisted of six trials. We broke down each set further by cross-folding them into five training and testing pairs of ratio 80% to 20%, respectively. While normally we do not split validation data, in this experiment, we are interested in both applying the fit and optimized algorithm, as well as identifying if the algorithm can be retrained on new data. We created each of the eight trials from a population size of 128 individuals run for 300 generations. The data comprises information about each crossover that occurred in each trial, including the descriptive features of the crossover that Section A.1 enumerated, as well as the supervised information as to whether or not the crossover was successful, according to Code Listing 5.2.

We extracted two non-dominated solutions from the optimization on the development data. One was a balanced solution, meaning the false positive rates and false negative weights were nearly equal; the other was a false negative optimized solution, meaning it had one of the lowest false negative rates without being a trivial solution. We hypothesized that driving the false negative rate to be as low as possible while suppressing as many false positives as permissible would lead to the best performance of the matchmaker in terms of

Table A.2: Balanced individual test scores

Fold	Accuracy	FP Rate	FN Rate	NFP	NFN	Naïve Accuracy
Fold 1	72.44%	27.77%	25.14%	6328	533	7.77%
Fold 2	73.27%	26.43%	30.38%	6341	520	7.58%
Fold 3	73.14%	26.72%	28.60%	6326	535	7.80%
Fold 4	72.32%	27.79%	26.31%	6344	517	7.54%
Fold 5	72.80%	27.52%	22.97%	6369	492	7.17%

Table A.3: False negative optimized individual test scores

Fold	Accuracy	FP Rate	FN Rate	NFP	NFN	Naïve Accuracy
Fold 1	27.07%	78.86%	2.63%	6328	533	7.77%
Fold 2	26.41%	79.40%	2.69%	6341	520	7.58%
Fold 3	25.46%	80.73%	1.31%	6326	535	7.80%
Fold 4	26.03%	79.85%	1.74%	6344	517	7.54%
Fold 5	26.42%	79.13%	1.63%	6369	492	7.17%

fastest and greatest AUC reduction. We constructed this hypothesis based on the idea that a false negative is more dangerous than a false positive for the optimization, i.e. suppressing an algorithm that would have been a successful exploration of the search space is more detrimental to performance than allowing the evaluation of something that would not help the evolution.

Tables A.2 and A.3 show the results of applying the balanced and false negative optimized algorithms, respectively, that EMADE produced on the testing partition of the development data during the optimization process. On the five folds of this dataset, the average accuracy of a naïve algorithm of assuming every crossover is successful is 7.8%. The naïve solution is, in effect, the traditional mating strategy, i.e. randomly pair all individuals. This accuracy is equivalent to the occurrence rate of successful crossovers out of all crossovers. In comparison with the naïve solution, both the balanced and false negative optimized algorithms greatly improve upon the accuracy achieved by the traditional approach.

Of course, to understand the performance of algorithms, we need to apply them to validation data not presented to them during the optimization. Performance on the validation data shows whether or not the algorithms are overtrained. Tables A.4 and A.5 show the results of applying the balanced and false negative algorithms EMADE produced, respec-

Table A.4: Balanced individual validation scores

Fold	Accuracy	FP Rate	FN Rate	NFP	NFN	Naïve Accuracy
Fold 1	59.95%	39.75%	41.98%	17756	2780	13.54%
Fold 2	60.91%	38.59%	42.26%	17727	2809	13.68%
Fold 3	57.95%	42.55%	38.83%	17773	2763	13.45%
Fold 4	60.97%	38.44%	42.67%	17670	2866	13.96%
Fold 5	57.57%	42.87%	39.73%	17692	2844	13.85%

tively, on the testing partition of the validation data. We partitioned this data so that we could refit our models on data from the same trials to understand the abstractness of the algorithms themselves. Performance significantly degrades on both individuals. However, the objective scores for the balanced individual remain balanced, and the objective scores for the false negative optimized individual remain strongly favoring false negatives. On the validation trials, the naïve classification performs nearly twice as well on the validation data than on the development data. This means that the validation data has a rate of success that is nearly twice that of the development data.

The high variability from trial to trial indicates several issues with the matchmaking algorithm. First, the variability indicates that it is difficult to use a model trained on one set of trials and apply it to another. This lack of transferability means that we cannot train a matchmaker using a feature set chosen for past data and apply to future optimizations. Second, the variability shows that not only are the the mappings from feature to rates of success inconsistent, but from trial to trial, the rates of success themselves can be drastically different. These wide differences between trials means that applying the concepts of crossover improvement from Section 5.2 is difficult, because the success rate can rise or fall so dramatically we cannot know *apriori* how much better or worse our matched crossover will be.

Since our trained matchmaking algorithms did not have transferable results, we investigated whether the untrained algorithms could be useful for online learning during a new optimization. To simulate this *in situ* retraining, we refit the balanced and false negative optimized algorithms on the training portion of the validation data. Tables A.6 and A.7

Table A.5: False negative optimized individual validation scores

Fold	Accuracy	FP Rate	FN Rate	NFP	NFN	Naïve Accuracy
Fold 1	23.25%	86.99%	11.33%	17756	2780	13.54%
Fold 2	24.17%	86.00%	11.64%	17727	2809	13.68%
Fold 3	21.07%	89.86%	8.61%	17773	2763	13.45%
Fold 4	24.07%	86.55%	10.43%	17670	2866	13.96%
Fold 5	22.67%	88.06%	10.55%	17692	2844	13.85%

Table A.6: Balanced individual retrained on validation scores

Fold	Accuracy	FP Rate	FN Rate	NFP	NFN	Naïve Accuracy
Fold 1	63.68%	35.44%	41.98%	17756	2780	13.54%
Fold 2	62.89%	36.86%	38.66%	17727	2809	13.68%
Fold 3	62.33%	37.51%	38.69%	17773	2763	13.45%
Fold 4	64.01%	34.80%	43.30%	17670	2866	13.96%
Fold 5	63.90%	35.26%	41.35%	17692	2844	13.85%

show the results for retraining the balanced and false negative algorithms, respectively, and scoring them on the testing portion of the validation data. Unfortunately, this leads to unpredictable results where our balanced algorithm now slightly favors reducing false positives. The false negative optimized algorithm yields results that vary wildly from fold to fold, which is an extremely undesirable characteristic for a machine learning algorithm. The conclusion we draw from this poor refitting performance is that extracted algorithm pipelines, developed on the chosen feature set and success criteria on one trial, are not useful for retraining *in situ* on another trial.

Table A.7: False negative optimized retrained on validation scores

Fold	Accuracy	FP Rate	FN Rate	NFP	NFN	Naïve Accuracy
Fold 1	74.96%	16.85%	77.37%	17756	2780	13.54%
Fold 2	20.92%	90.68%	5.84%	17727	2809	13.68%
Fold 3	40.53%	62.34%	41.01%	17773	2763	13.45%
Fold 4	86.55%	1.23%	88.80%	17670	2866	13.96%
Fold 5	27.88%	81.21%	15.54%	17692	2844	13.85%

A.5 Conclusions and Future Work

Section 5.2 showed that we can quantify an expected improvement in the area under the curve during an optimization if we could understand an increase in probability of success of an individual crossover. However, when we applied this to a strategic mating operator called a matchmaker, we saw that we could not affect positive change during an optimization. Diving deeper into the problem using EMADE, we saw that the matchmaker could improve performance drastically against the naïve approach of assuming all pairings produce successful crossover. However, these improvements were only available on the trial data for which we optimized the algorithms. The trained models were not applicable to new trials, and their performance was not consistent when retrained on new data. Future work could explore the concept of a co-evolution paradigm, where matchmakers are evolved during an optimization, allowing machine learning algorithms with low false negative rates to be evolved to the uniqueness of the evolutionary trial at hand.

REFERENCES

[1] J. H. Holland, *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975.

[2] C. Darwin, "On the origins of species by means of natural selection," *London: Murray*, 1859.

[3] J. R. Koza, "Survey of genetic algorithms and genetic programming," in *WESCON/'95. Conference record.'Microelectronics Communications Technology Producing Quality Products Mobile and Portable Power Emerging Technologies'*, IEEE, 1995, p. 589.

[4] L. Davis, "Genetic algorithms and simulated annealing," 1987.

[5] D. J. Montana, "Strongly typed genetic programming," *Evolutionary computation*, vol. 3, no. 2, pp. 199–230, 1995.

[6] J. R. Koza, *Genetic Programming: vol. 1, On the programming of computers by means of natural selection*. MIT press, 1992, vol. 1.

[7] C. M. Fonseca and P. J. Fleming, "An overview of evolutionary algorithms in multiobjective optimization," *Evolutionary computation*, vol. 3, no. 1, pp. 1–16, 1995.

[8] N. Srinivas and K. Deb, "Muiltiobjective optimization using nondominated sorting in genetic algorithms," *Evolutionary computation*, vol. 2, no. 3, pp. 221–248, 1994.

[9] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[10] D. Corne, J. Knowles, and M. Oates, "The Pareto envelope-based selection algorithm for multiobjective optimization," ... *Problem Solving from Nature PPSN VI*, no. Mcdm, 2000.

[11] D. Corne and N. Jerram, "PESA-II: Region-based selection in evolutionary multiobjective optimization," *Proceedings of the ...*, 2001.

[12] E. Zitzler, M. Laumanns, and L. Thiele, "Spea2: Improving the strength pareto evolutionary algorithm," 2001.

[13] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach," *Evolutionary Computation, IEEE Transactions on*, vol. 3, no. 4, pp. 257–271, 1999.

[14] J. Lehman and K. O. Stanley, "Exploiting Open-Endedness to Solve Problems Through the Search for Novelty," *Artificial Life XI*, pp. 329–336, 2008.

[15] Q. Zhang and H. Li, "Moea/d: A multiobjective evolutionary algorithm based on decomposition," *IEEE Transactions on evolutionary computation*, vol. 11, no. 6, pp. 712–731, 2007.

[16] N. Beume, B. Naujoks, and M. Emmerich, "SMS-EMOA: Multiobjective selection based on dominated hypervolume," *European Journal of Operational Research*, vol. 181, no. 3, pp. 1653–1669, 2007.

[17] J. Bader and E. Zitzler, "HypE: an algorithm for fast hypervolume-based many-objective optimization.," *Evolutionary computation*, vol. 19, no. 1, pp. 45–76, 2011.

[18] G. Rohling, "Multiple objective evolutionary algorithms for independent, computationally expensive objective evaluations," PhD thesis, Georgia Institute of Technology, 2004.

[19] K. Dolan, *Genetic programming source*.

[20] L. Beadle and C. G. Johnson, "Semantically driven crossover in genetic programming," in *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, IEEE, 2008, pp. 111–116.

[21] H. Ishibuchi and Y. Shibata, "Mating Scheme for Controlling the Diversity-Convergence Balance for Multiobjective Optimization," in *Genetic and Evolutionary Computation–GECCO*, Springer Berlin Heidelberg, 2004, pp. 1259–1271.

[22] H. Guo, L. Jack, and A. Nandi, "Feature generation using genetic programming with application to fault classification," *Systems, Man, and Cybernetics, ...*, vol. 35, no. 1, pp. 89–99, 2005.

[23] K. Holladay and K. Robbins, "Evolution of signal processing algorithms using vector based genetic programming," *Digital Signal Processing, 2007 ...*, pp. 2–5, 2007.

[24] C. D. Vera, E. Alfaro-cid, K. Sharman, and A. I. Esparcia-alc, "Genetic Programming and Serial Processing," vol. 22, no. 2, pp. 265–285, 2014.

[25] J. Streater, "Genetic Programming for the Automatic Construction of Features in Skin-Lesion Image Classification," *inf.ed.ac.uk*, 2010.

[26] L. Guo, D. Rivero, J. Dorado, C. R. Munteanu, and A. Pazos, "Automatic feature extraction using genetic programming: An application to epileptic eeg classification," *Expert Systems with Applications*, vol. 38, no. 8, pp. 10 425–10 436, 2011.

[27] H. Al-Sahaf, A. Song, and M. Zhang, "Hybridisation of Genetic Programming and Nearest Neighbour for classification," in *2013 IEEE Congress on Evolutionary Computation*, IEEE, Jun. 2013, pp. 2650–2657, ISBN: 978-1-4799-0454-9.

[28] K. Bache and M. Lichman, *UCI machine learning repository*, 2013.

[29] Y.-S. Lee and L.-I. Tong, "Forecasting time series using a methodology based on autoregressive integrated moving average and genetic programming," *Knowledge-Based Systems*, vol. 24, no. 1, pp. 66–72, Feb. 2011.

[30] P. Ravisankar, V. Ravi, and I. Bose, "Failure prediction of dotcom companies using neural network–genetic programming hybrids," *Information Sciences*, vol. 180, no. 8, pp. 1257–1267, Apr. 2010.

[31] D. Zhang, M. Hifi, Q. Chen, and W. Ye, "A Hybrid Credit Scoring Model Based on Genetic Programming and Support Vector Machines," in *2008 Fourth International Conference on Natural Computation*, Ieee, 2008, pp. 8–12, ISBN: 978-0-7695-3304-9.

[32] J. Sherrah, R. Bogner, and A. Bouzerdoum, "The evolutionary pre-processor: Automatic feature extraction for supervised classification using genetic programming," *Genetic Programming*, 1997.

[33] R. Hassan and B. Cohanim, "A comparison of particle swarm optimization and the genetic algorithm," *... design optimization ...*, pp. 1–13, 2005.

[34] S. S. Skiena, *The algorithm design manual: Text*. Springer Science & Business Media, 1998, vol. 1.

[35] P. V. Laarhoven and E. Aarts, *Simulated annealing*. Springer Netherlands, 1987.

[36] R. S. Olson, R. J. Urbanowicz, P. C. Andrews, N. A. Lavender, L. C. Kidd, and J. H. Moore, "Applications of evolutionary computation: 19th european conference, evoapplications 2016, porto, portugal, march 30 – april 1, 2016, proceedings, part i," in, G. Squillero and P. Burelli, Eds. Springer International Publishing, 2016, ch. Automating Biomedical Data Science Through Tree-Based Pipeline Optimization, pp. 123–137, ISBN: 978-3-319-31204-0.

[37] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Courna-

peau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[38] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, 2012.

[39] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms," in *Proc. of KDD-2013*, 2013, pp. 847–855.

[40] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., Curran Associates, Inc., 2015, pp. 2962–2970.

[41] A. Klein, S. Falkner, N. Mansur, and F. Hutter, "Robo: A flexible and robust bayesian optimization framework in python," in *NIPS 2017 Bayesian Optimization Workshop*, Dec. 2017.

[42] *H2o.ai*, `https://www.h2o.ai`, Accessed: 2018-09-22.

[43] *Rapidminer*, `https://rapidminer.com`, Accessed: 2018-09-22.

[44] *Datarobot*, `https://www.datarobot.com`, Accessed: 2018-09-22.

[45] *Google cloud automl*, `https://cloud.google.com/automl/`, Accessed: 2018-09-22.

[46] F. Chollet *et al.*, *Keras*, `https://keras.io`, 2015.

[47] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *CoRR*, vol. abs/1603.02754, 2016. arXiv: `1603.02754`.

[48] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., Curran Associates, Inc., 2017, pp. 3146–3154.

[49] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[50] M O'neill, C Ryan, M Keijzer, and M Cattolico, "Crossover in grammatical evolution," *Genetic programming and ...*, pp. 67–93, 2003.

[51] Wikipedia, *Occam's razor — Wikipedia, the free encyclopedia*, [Online; accessed 15-February-2015], 2015.

[52] F. Azadivar and J. Wang, "Facility layout optimization using simulation and genetic algorithms," *International Journal of Production Research*, vol. 38, no. 17, pp. 4369–4383, Nov. 2000.

[53] Y. Hold-Geoffroy, O. Gagnon, and M. Parizeau, "Once you scoop, no need to fork," in *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment*, ACM, 2014, p. 60.

[54] N. Y. Hammerla, R. Kirkham, P. Andras, and T. Ploetz, "On preserving statistical characteristics of accelerometry data using their empirical cumulative distribution," in *Proceedings of the 2013 International Symposium on Wearable Computers*, ACM, 2013, pp. 65–68.

[55] S IHO, *44. iho standards for hydrographic surveys*, 2008.

[56] A. Mathur, V. Ramnath, V. Feygels, E. Fuchs, J. Y. Park, and G. H. Tuell, "Predicted lidar ranging accuracy for czmil," in *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XVI*, International Society for Optics and Photonics, vol. 7695, 2010, 76950Z.

[57] D. A. Carr, "A study of the target detection capabilities of an airborne lidar bathymetry system," PhD thesis, Georgia Institute of Technology, 2013.

[58] W. H. Richardson, "Bayesian-based iterative method of image restoration," *JOSA*, vol. 62, no. 1, pp. 55–59, 1972.

[59] L. B. Lucy, "An iterative technique for the rectification of observed distributions," *The astronomical journal*, vol. 79, p. 745, 1974.

[60] K. Guo, W. Xu, Y. Liu, X. He, and Z. Tian, "Gaussian half-wavelength progressive decomposition method for waveform processing of airborne laser bathymetry," *Remote Sensing*, vol. 10, no. 1, p. 35, 2017.

[61] M. Słota, "Decomposition techniques for full-waveform airborne laser scanning data," *Geomatics and Environmental Engineering*, vol. 8, 2014.

[62] J. Zhu, Z Zhang, X. Hu, and Z Li, "Analysis and application of lidar waveform data using a progressive waveform decomposition method," *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci.*, vol. 38, 2011.

[63] A Roncat, G Bergauer, and N Pfeifer, "Retrieval of the backscatter cross-section in full-waveform lidar data using b-splines," *Proc. Int. Archives Photogramm. Remote Sens. Spatial Inf. Sci*, pp. 137–142, 2010.

[64] A. Chauve, C. Mallet, F. Bretar, S. Durrieu, M. Pierrot-Deseilligny, and W. Puech, "Processing full-waveform lidar data: Modelling raw signals," in *International archives of photogrammetry, remote sensing and spatial information sciences 2007*, 2008, pp. 102–107.

[65] C. Mallet and F. Bretar, "Full-waveform topographic lidar: State-of-the-art," *ISPRS Journal of photogrammetry and remote sensing*, vol. 64, no. 1, pp. 1–16, 2009.

[66] K. Ding, Q. Li, J. Zhu, C. Wang, M. Guan, Z. Chen, C. Yang, Y. Cui, and J. Liao, "An improved quadrilateral fitting algorithm for the water column contribution in airborne bathymetric lidar waveforms," *Sensors*, vol. 18, no. 2, p. 552, 2018.

[67] L. Abady, J.-S. Bailly, N. Baghdadi, Y. Pastol, and H. Abdallah, "Assessment of quadrilateral fitting of the water column contribution in lidar waveforms on bathymetry estimates," *IEEE Geoscience and Remote Sensing Letters*, vol. 11, no. 4, pp. 813–817, 2014.

[68] R. K. McConnell, *Method of and apparatus for pattern recognition*, US Patent 4,567,610, 1986.

[69] E. Tola, V. Lepetit, and P. Fua, "A fast local descriptor for dense matching," 2008.

[70] R. M. Haralick, K. Shanmugam, *et al.*, "Textural features for image classification," *IEEE Transactions on systems, man, and cybernetics*, no. 6, pp. 610–621, 1973.

[71] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European conference on computer vision*, Springer, 2016, pp. 630–645.

[72] J. Zutty, D. Long, H. Adams, G. Bennett, and C. Baxter, "Multiple objective vector-based genetic programming using human-derived primitives," in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, ACM, 2015, pp. 1127–1134.

[73] J. Zutty, D. Long, and G. Rohling, "Increasing the throughput of expensive evaluations through a vector based genetic programming framework," in *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, ACM, 2016, pp. 1477–1478.

[74] R. S. Olson, N. Bartley, R. J. Urbanowicz, and J. H. Moore, "Evaluation of a tree-based pipeline optimization tool for automating data science," in *Proceedings of the*

2016 on Genetic and Evolutionary Computation Conference, ACM, 2016, pp. 485–492.

[75] R. S. Olson and J. H. Moore, "Identifying and harnessing the building blocks of machine learning pipelines for sensible initialization of a data science automation tool," *arXiv preprint arXiv:1607.08878*, 2016.

[76] P. Liskowski and K. Krawiec, "Online discovery of search objectives for test-based problems," *Evolutionary computation*, 2016.

[77] ——, "Non-negative matrix factorization for unsupervised derivation of search objectives in genetic programming," in *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*, ACM, 2016, pp. 749–756.

[78] O. Ledoit and M. Wolf, "A well-conditioned estimator for large-dimensional covariance matrices," *Journal of multivariate analysis*, vol. 88, no. 2, pp. 365–411, 2004.

[79] S. Wilhelm, "Gibbs sampler for the truncated multivariate normal distribution," 2015.

[80] A. Reiss and D. Stricker, "Creating and benchmarking a new dataset for physical activity monitoring," in *Proceedings of the 5th International Conference on PErvasive Technologies Related to Assistive Environments*, ACM, 2012, p. 40.

[81] A. Baldominos, Y. Saez, and P. Isasi, "Feature set optimization for physical activity recognition using genetic algorithms," in *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, ACM, 2015, pp. 1311–1318.

[82] O. Rudenko and M. Schoenauer, "Dominance based crossover operator for evolutionary multi-objective algorithms," in *International Conference on Parallel Problem Solving from Nature*, Springer, 2004, pp. 812–821.

[83] E. Zitzler and L. Thiele, "Multiobjective optimization using evolutionary algorithms-a comparative case study," in *international conference on parallel problem solving from nature*, Springer, 1998, pp. 292–301.

[84] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. Da Fonseca, "Performance assessment of multiobjective optimizers: An analysis and review," *IEEE Transactions on evolutionary computation*, vol. 7, no. 2, pp. 117–132, 2003.

[85] J. Derrac, S. García, D. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 3–18, 2011.

[86] S. García, A. Fernández, J. Luengo, and F. Herrera, "Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power," *Information Sciences*, vol. 180, no. 10, pp. 2044–2064, 2010.

[87] M. M. Ali, C. Khompatraporn, and Z. B. Zabinsky, "A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems," *Journal of global optimization*, vol. 31, no. 4, pp. 635–672, 2005.

[88] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, pp. 50–60, 1947.

[89] N. Q. Uy, N. X. Hoai, M. O'Neill, and B. McKay, "The role of syntactic and semantic locality of crossover in genetic programming," in *International Conference on Parallel Problem Solving from Nature*, Springer, 2010, pp. 533–542.

[90] K. Zhang and D. Shasha, "Simple fast algorithms for the editing distance between trees and related problems," *SIAM journal on computing*, vol. 18, no. 6, pp. 1245–1262, 1989.

[91] J. Bacardit and N. Krasnogor, *The icos psp benchmarks repository*, `http://icos.cs.nott.ac.uk/datasets/psp_benchmark.html`, 2008.

VITA

Jason Zutty came to Georgia Tech in 2005 to pursue his Bachelor's degree in Electrical Engineering, which he obtained in 2009. He continued to achieve his Master's in Electrical Engineering in 2010. Further interested in signal processing, he decided to begin work on a Ph.D. in the field. He joined the Quantitative Ultrasonic Evaluation, Sensing and Testing (QUEST) Laboratory as a graduate research assistant, before finding his way to the Electro-Optical Systems Laboratory (EOSL) at the Georgia Tech Research Institute (GTRI). After finishing his coursework, he began working full time at GTRI-EOSL in 2012 in the field of automated machine learning. At GTRI-EOSL, he developed the Evolutionary Multi-objective Algorithm Design Engine and completed his research.