ACTIVE LEARNING IN GENETIC PROGRAMMING

By

Nathan Haut

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Computational Mathematics, Science, and Engineering—Doctor of Philosophy

2023

ABSTRACT

Active learning is an active field within machine learning that aims to minimize the amount of training data required by focusing on selecting training points that will be maximally informative for model development. Active learning has been successfully applied to many different types of machine learning, but until this dissertation, active learning's application to genetic programming has not been thoroughly examined. Considering that genetic programming is already known to be less data-hungry than many other methods, it seemed to be natural that we could further reduce training data requirements for genetic programming by applying active learning methods.

In this dissertation, I developed the active learning in genetic programming (AL-GP) method and demonstrated how it is flexible and can be applied to a diverse set of population-based machine learning systems across several problem domains to guide training data selection. This results in a reduction in training data required to arrive at high-quality models. The method is shown to be effective across regression, image segmentation, and image classification problems.

For active learning in regressions tasks, I explored the impact of both model uncertainty and data diversity individually and together. For image analysis tasks, I explored the impact that ensemble diversity has on active learning success.

In this work, I developed two new GP systems, StackGP and DT-GP. Additionally, I modified the existing SEE-Segment system to improve the search strategy. The AL-GP approach was shown to work with all three systems which demonstrates that the AL-GP approach is general and easy to adapt to any population-based machine learning system.

Although not directly linked to active learning but key to the success of StackGP for regression tasks, correlation as a fitness function was shown to be more effective than the traditional RMSE fitness function.

ACKNOWLEDGMENTS

Dr. Bill Punch and Dr. Wolfgang Banzhaf co-advised me while a Ph.D. student at MSU and offered valuable insight and guidance that was critical to my success in research and the completion of this dissertation. Dr. Mark Kotanchek first introduced me to the world of genetic programming and taught me valuable skills that contributed significantly to the accelerated timeline of my Ph.D. research. He inspired me to pursue research at a graduate level so that I could challenge myself and gain valuable skills that will hopefully allow me to make a meaningful contribution to this world. Dr. Dirk Colbry welcomed me into his research group where an ideal synergy was realized that allowed me to learn about and apply my work to new domains while also bringing knowledge of genetic programming and active learning to their group and new functionality to their SEE-Segment tool.

Significant computing resources were supplied by MSU's iCER high-performance computing center, which allowed me to complete all of my experiments for this dissertation which required computing hours in the range of 7-figures to be completed in a timespan of just about three years.

Most importantly, I thank Brianna Ricker, whose unconditional support helped me make it through my graduate studies. Thank you for being patient with me through all the times I was hopelessly trying to fix bugs in my code at 2am or I was raving excitedly about a new idea I had for my research. As well, thank you for offering your artistic eye anytime I was struggling to design a poster, presentation, or figure.

TABLE OF CONTENTS

CHAPTER 1

CONTRIBUTIONS

In this dissertation, I developed the active learning in genetic programming (AL-GP) approach and demonstrated how it is flexible and can be applied to a diverse set of population-based machine learning systems across several problem domains. In pursuit of developing the AL-GP method, I explored how the approach needs to be adapted to work with regression, classification, and image segmentation tasks.

For regression tasks, I explored the importance of data diversity and model uncertainty and compared several metrics for computing each. This is shown in Chapter 6. I demonstrate that differential entropy is the clear winner for computing uncertainty, while correlation and Euclidean distance each have unique benefits when used to measure data diversity. I also developed an approach to consider both data diversity and model uncertainty together for active learning by using a Pareto optimization approach.

In Chapter 9, I applied AL-GP to image segmentation and classification tasks. I show that AL-GP can reduce the number of images or pixels required for training models in image analysis problems. I also explore how AL-GP can be applied to a research workflow to accelerate a scientific research project that requires segmenting and classifying biological cells.

A key step in developing the AL-GP method was identifying that the model population present in genetic programming systems can be exploited by capturing the diversity of the high-quality individuals in the population and using their disagreement as the metric for uncertainty to select informative training samples. For regression tasks, I developed an ensemble selection method that considers high-quality candidate models for different training clusters, to capture the diverse models that perform well in different regions of the training space. For classification and segmentation tasks, I compare two approaches for generating model ensembles and demonstrate that a more strict definition for diversity when selecting ensembles leads to better performance of active learning.

To test the validity of the AL-GP approach, I had to develop several new GP systems as part of this dissertation. I developed two new GP systems, StackGP and DT-GP, discussed in Chapters

4 and 7. StackGP is a stack-based genetic programming system that pulls from other existing GP systems to produce a state-of-the-art system for symbolic regression tasks. The key features of this system are the stack-based model structures, correlation fitness function paired with an essential alignment step, multi-objective search strategy via Pareto tournament selection, and the data clustering ensemble generation method. DT-GP is a GP system that is specialized to produce decision trees. It is unique in the sense that the model structure is hierarchical and typed, so that all models produced are valid, which improves search efficiency. As well, a relative fitness measure was used that considers models in both their raw form and their form if the Not operator was applied, which effectively reduces the size of the search space. AL-GP was also applied to SEE-Segment, which I did not create, but did modify to improve the performance of the search so that active learning could be effective. SEE-Segment and the modifications I made are discussed in Chapter 8.

One of the key features responsible for StackGP's performance is the use of correlation as a fitness function when it is paired with an alignment step. This implementation choice had not been thoroughly explored or explained in the literature prior to this work, so Chapter 5 provides a detailed analysis of the benefits of using correlation as a fitness function with a simple linear regression alignment step compared to the traditional RMSE fitness function. This also identifies the potential benefits of developing relative fitness measures for other problem domains when using GP to help improve the efficiency of the evolutionary search by improving the visibility of solutions and shrinking the search space.

Part I

Background

Machine learning (ML) is the process of training a computer algorithm to perform a specific task through some learning process. There are three main learning strategies that exist in machine learning: supervised learning, unsupervised learning, and reinforcement learning. Supervised learning is applied when you have a set of training data and each training case has a label, the learning process is an optimization of a model so that the model most accurately maps the training input to the training labels. Unsupervised learning also has a set of training data, except here the data has no labels, so the goal is to build a model that discovers some pattern in the data. A common type of unsupervised learning is clustering, where the goal is to discover groups in the data, where within each group samples are similar in some way, and samples within different groups differ in some meaningful way from samples in other groups. Reinforcement learning is a bit different, where the goal instead is to learn by actually attempting to perform a task and then the algorithm is either rewarded or penalized based on how the algorithm performs. Through many iterations, reinforcement learning will lead to improved performance on the specific task. A common application of reinforcement learning is in robotics. For example, you might want to train a robot vacuum to clean a space. The robot would be rewarded as it covers more of the floor space and it would be penalized when it repeats spaces or runs into obstacles. In this work, we are focused on both supervised and unsupervised learning. We will use supervised learning to train models since all models will be trained on training data with labels. Unsupervised learning will be used to analyze unlabelled training points to determine how informative the points might be if we gave them a label and supplied them to the model's training set. This will be described in more detail throughout this dissertation.

There are a plethora of model representations that are suited for different types of tasks. For each representation, the strategy for learning has to be developed to best suit that representation. Common representations include support vector machines (SVMs) [Vapnik, 2006], neural networks (NNs) [Müller et al., 1995], random forests (RFs) [Ho, 1995], decision trees (DTs) [Quinlan, 1986], Gaussian mixture models (GMMs) [Rasmussen, 1999], regression models, etc. Support vector machines are well suited for finding decision boundaries between multiple classes of samples, thus

it is applied to classification problems. The learning process involves finding a decision boundary that maximizes the classification accuracy of the training samples. Regression models are a class of models which contain mathematical equations. The goal during learning is to find an equation or adjust parameters within an equation to best fit the training data labels when supplied with the training data input values. In this work, we will focus on regression models, decision trees, and also a unique representation for image segmentors.

Depending on the type of model, there are different methods of training that can be used. Often, a single model of a specific form will be defined. To train that model, an optimizer will be used with the model to adjust model parameters iteratively in search of a parameter set that allows the model to perform best on the training set. For example, when using a neural network, it is common to use backpropagation to adjust the weights of the model to improve performance on the training set. Alternatively, instead of starting with a single model, you can use a population of models and allow them to compete and adapt over many iterations, such that the performance of the models in the population improves over iterations. This method is called evolutionary computation and is inspired by biological evolution. For this work, I focus on evolutionary computation methods where we have a population of models that are evolving to solve a specific problem.

CHAPTER 2

ACTIVE LEARNING

Active learning [Cohn et al., 1996] is an iterative machine learning strategy where the ML method itself selects additional training data to maximally inform its own learning process. New points are selected using some metric of informativeness or uncertainty over the potential new training points. The method of determining which point will be maximally informative is dependent on the type of machine learning model being used, as well as the type of data. In general, there are three main classifications of active learning: pool-based, stream-based, and membership query synthesis [Ren et al., 2021]. Figure 2.1 shows a simple visual representation to compare the three methods of active learning. Pool-based active learning methods begin with a set of training data that is mostly unlabelled. The machine learning method is trained on the labelled data and then the uncertainty or informativeness metric is used to select one or several of the unlabelled points that rank highest according to the predicted amount of information that will be gained by labelling each point. Those points are then labelled and added to the training data. Stream-based methods are similar to pool-based methods except they are computationally lighter since instead of ranking all unlabelled data points, they consider only one point at a time and determine if it either should or should not be labelled. Membership query synthesis methods differ from the other two methods since they do not have a set of unlabelled points to choose from. Instead, the machine learning model is responsible for generating a point that the model's informativeness measure indicates will give relatively high information gain if produced and labelled. All three methods share the same goal. They attempt to minimize the total number of labelled data points needed to train a model of sufficient quality.

Active learning in the field of machine learning drew its inspiration from the statistics method of query learning. The goal of query learning is to design an experiment that will lead to maximal information gain [Lindley, 1956], similar to active learning, except the goal is not specific to improving a model directly, just gaining more knowledge. It views information from a statistical viewpoint, such that information gain can be quantified. This allows the information space to be

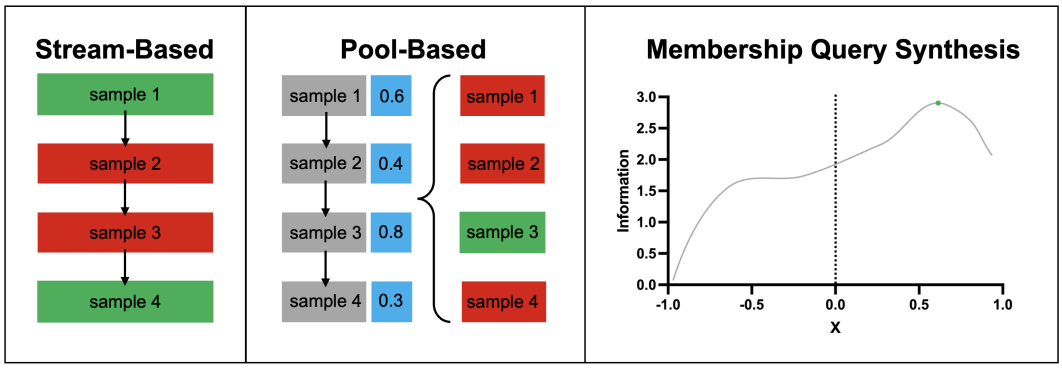


Figure 2.1 The three main types of active learning: Stream-based, pool-based, and membership query synthesis are visually demonstrated. Stream-based approaches, shown on the left, search through the samples one at a time and either mark them for labelling or skip them. Green indicates a sample is found to be informative and is marked for labelling, red indicates a sample is skipped. Pool-based approaches, shown in the middle, assign an information score to each potential training sample, and the most informative sample is chosen to be labelled and added to the training set. Membership query synthesis, shown on the right, searches a space of potential points not yet collected while maximizing an information measure and selects a point to be synthesized and labelled that maximizes the information score. The selected point is indicated by the green circle, while the y-axis of the curve represents the informativeness measure and the x-axis is representative of the sample space.

explored, with respect to previous knowledge, to find an experiment that would provide maximal information gain. In that work, Lindley provides a definition for average information gained from experiment ϵ when having prior knowledge $p(\theta)$:

$$g(\epsilon, p(\theta)) = E_x(g_1(x) - g_0); \tag{2.1}$$

$$g_0 = \int p(\theta) * log(p(\theta)) * d\theta; \tag{2.2}$$

$$g_1(x) = \int p(\theta|x) * log(p(\theta|x)) * d\theta \tag{2.3}$$

where x is the observed value that results from the experiment ϵ and θ represents the experimental parameters or the state of the system. This measure of experimental information gain is useful when there exists prior knowledge and the goal is to directly compare the amount of information gained by two or more experiments.

Uncertainty sampling is an approach to active learning that quantifies information gain using an uncertainty metric [Lewis and Gale, 1994]. It relies on having some metric to compute uncertainty

given a specific model. This was first used in [Lewis and Gale, 1994] for the tasks of selecting training samples for text classification of news headlines. It was shown that uncertainty sampling led to improved performance when compared to random sampling and relevance sampling. Relevance sampling is essentially the opposite of uncertainty sampling, where new samples are selected that are *most* likely to belong to a class. As well, it was shown that the models developed when using uncertainty sampling performed better when trained on a small data set than models trained on all samples. Sampling bias, which results from sampling from an unbalanced data set was shown to be an issue with random sampling, since few cases in under-represented samples are generally selected, which results in biases in the trained classifiers. Uncertainty sampling has the potential to reduce sampling bias since cases in the under-represented classes will be identified as interesting if a proper uncertainty metric is utilized.

As mentioned previously, the method for determining which points will be maximally informative depends on the machine learning method being used. For example, active learning has been applied to support vector machines. Support vector machines are linear discriminators, where the goal of learning is to find a decision boundary that both maximizes the number of correctly classified cases, while also maximizing the distance of the decision boundary to the training points in different classes. A simple uncertainty sampling based implementation for active learning was performed by computing the distance of all points to the separating hyperplane and selecting the point nearest the hyperplane to be labelled [Kremer et al., 2014]. For neural networks, one method that has been applied is to select points with the minimum difference between the predicted probabilities of the two most probable labels [Ren et al., 2021]. This can be defined as $M = P(l1|x) - P(l2|x)$, where M is the margin between the two most probable labels, $l1$ is the most probably label for input x, and $l2$ is the second most probable label for input x [Ren et al., 2021]. There are many other active learning methods even just within neural networks and support vector machine applications, but these are just two intuitive examples that highlight how the methods differ depending on the type of machine learning method being used.

Uncertainty cannot be computed on all model forms directly. In such cases, it is possible to

use several models from a set and use disagreement as the metric for uncertainty. An instance of this approach has used the version space, which is the set of models that fit all the training samples [Mitchell, 1982]. Query-by-committee is an active learning method that has been utilized for classification tasks, where a sample of models from the version space is selected and new training samples are selected from the set of samples where those models disagree [Seung et al., 1992]. They demonstrated that information was gained much faster when using query-by-committee when compared to random sampling on two toy problems.

Active learning using query-by-committee has been applied to genetic programming classification tasks where a committee of models votes on the class of data pairs, and points are only labelled when the committee of developing models encounters pairs that can't be classified [De Freitas et al., 2010]. This was found to reduce the total effort needed to label training points, since only a subset had to be labelled before finding accurate models. Active learning has also been applied to genetic programming where training sets are large, by selecting sub-samples of the training data to be used [Curry et al., 2007, Lasarczyk et al., 2004]. In [Curry et al., 2007], active learning was performed by segmenting the data into smaller blocks and training the models using one block at a time randomly selected with uniform probability. As training continues, bias is introduced into the probability by increasing the bias towards blocks that haven't been seen in a while as well as blocks where the models performed poorly during training. Active learning for sub-sampling with genetic programming was found to decrease training times to find quality binary classification models by an order of magnitude [Curry et al., 2007]. In [Lasarczyk et al., 2004], subsets were selected by dynamically developing a fitness case topology that could be used to create minimally related subsets of data. In this context, the strength of a relationship between two training cases was indicated by the number of individuals that were able to solve both training cases. Active learning has also been applied to the task of discovering regular expressions using genetic programming [Bartoli et al., 2018]. In that work, they used a restricted query-by-committee (rQbC) strategy that utilized the top 25% of models in a population to generate "extraction queries", in which the user then indicates whether or not the character string selected by the "extraction query" should be

extracted or not by a regular expression.

Active learning methods for machine learning have shown to be very successful in applied settings to improve the method of labelling and collecting data with various machine learning types. Active learning has recently been demonstrated to significantly reduce the labelling efforts required for labelling data associated with identifying heart disease [El-Hasnony et al., 2022]. They demonstrated that they could find more accurate models using fewer data points when compared to a random point selection strategy. In the realm of cybersecurity, active learning with support vector machines has been applied to decrease the number of training samples needed to develop models to identify PDF documents that have been infected with malware [Li et al., 2022]. They demonstrated that they could reduce the required training set size to $1/30th$ of the required size when compared to an approach without active learning while maintaining model performance. Active learning methods using an uncertainty measure with neural networks were utilized within chemistry to effectively explore a space of 16 million potential catalysts to maximize the conversion rate of methane to methanol [Nandy et al., 2022]. The researchers discussed that this active learning strategy could be applied to other chemistry problems with large search spaces to reduce decade-scale research projects down to timescales of years or weeks.

While active learning has been primarily applied to supervised learning since the goal is generally to generate labelled data, there has been some recent work in applying active learning methods to unsupervised learning and reinforcement learning, the other two classes of machine learning. Active learning approaches for unsupervised learning differ from active learning approaches for supervised learning. For unsupervised learning, rather than selecting maximally informative points for *labelling*, points are selected that are maximally *representative* of the sample space without consideration for labels [Li et al., 2015]. In [Yu et al., 2006] they propose Transductive Experimental Design (TED), which is a method for choosing points that are maximally representative of the sample space. They demonstrate the use of this approach on regression tasks and suggest it could be extended to classification tasks. Active learning in unsupervised scenarios has also been shown to be useful to sample maximally informative points in the early stages of an active learning process

prior to any points existing in the training set [Nie et al., 2013]. This can be used to collect an initial set of representative data points before transitioning to a supervised active learning strategy. As well, since the focus is on representative samples, outliers do not negatively impact learning.

Active learning in reinforcement learning seems to be the least explored area of active learning, but there have been a few examples of researchers trying to apply active learning to reinforcement learning methods. In [Doshi-Velez et al., 2012], researchers apply an active learning method to be used with partially observable Markov decision processes (POMDPs). There they allow a reinforcement learning agent to query an expert on the correct or best policy when the agent encounters a scenario where the predicted costs of all known actions exceed the cost of querying an expert for the correct policy. Active learning for reinforcement learning has also been applied to minimize the number of samples required by querying for samples in specific states [Lopes et al., 2009]. The specific focus of that work was inverse reinforcement learning where the goal was to approximate a reward function using minimal data. As well, there has been work on using reinforcement learning to train an active learning agent that decides which information should be queried [Rudovic et al., 2019]. The focus of that work was to use reinforcement learning to develop an active learner that chooses informative samples to approximate human engagement.

In this dissertation, the active learning methods developed primarily take inspiration from uncertainty sampling, query-by-committee, and query learning while using pool-based and membership query synthesis active learning techniques. Uncertainty sampling with query-by-committee was used to develop model-driven active learning strategies for selecting training data and is explored extensively throughout this dissertation across various applications. For a data-driven active learning approach explored in Chapter 6, inspiration is taken from query learning to select informative training samples given the existing training sets for regression problems.

CHAPTER 3

EVOLUTIONARY COMPUTATION

Evolutionary computation is a field of machine learning that is inspired by natural evolution. Specifically, evolutionary computation simulates natural selection by determining the survival of individuals from a population based on a fitness function which is a metric that determines how well each individual accomplishes an assigned task. To allow for exploration and innovation, evolutionary computation methods utilize the genetic operators: mutation, crossover, and selection [Back and Schwefel, 1996]. Random mutation allows for new genetic information to be considered, making it the operator of innovation. Crossover allows genetic information to be exchanged between individuals in a population, potentially allowing beneficial genetic information from multiple individuals to come together to create a more fit individual. Selection determines which individuals in the population will have the chance to survive and reproduce to generate the next generation. The dynamics of evolution in evolutionary computation are primarily controlled by modifying the methods and rates of selection, mutation, and crossover. A general overview of an evolutionary computation algorithm is shown in Figure 3.1.

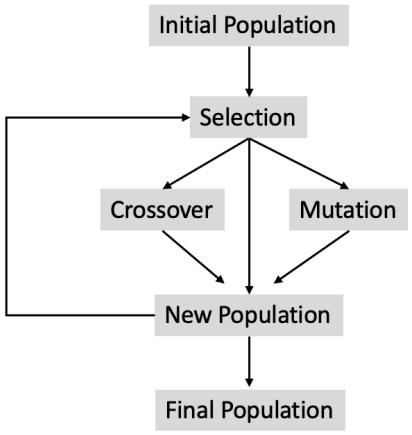


Figure 3.1 An generalized overview of how an evolutionary computation algorithm works. It begins with an initial population, typically randomly generated, and using a selection operator to determine which individuals survive and get to reproduce via crossover mutation, the next population is produced. This iterates many times until a termination condition is met.

There are many different methods for selecting which individuals from a population will survive

and reproduce. Random selection uses no selection pressure at all and allows any individual to be selected and to reproduce without any bias for how the individual performs. This isn't used in practice, but if it were used for optimization, this would be equivalent to random search. While random search is guaranteed to find a solution eventually, if one exists, it is not efficient so it is necessary to apply some selection pressure. Tournament selection is currently a commonly used method, where individuals are selected to compete in random tournaments of some set size, and the best individual in that tournament gets the opportunity to reproduce. Selection pressure in tournaments can be modified by adjusting the tournament size. A large tournament increases selection pressure while smaller tournaments weaken the selection pressure. More recently, a new variant of tournament selection has been developed, Pareto tournament selection, which can consider multiple fitness objectives simultaneously [Kotanchek et al., 2007]. Pareto tournament selection begins exactly as tournament selection, with a randomly selected group of individuals, but instead of always selecting one winner, all non-dominated individuals in the tournament are returned as winners and given the opportunity to reproduce. This idea of optimizing multiple objectives has recently been adapted into Lexicase selection, which works by randomly ordering fitness cases and objectives and having individuals compete on each fitness case until one individual wins [Spector, 2012]. This approach has recently been claimed to improve the preservation of specialists but is being actively explored to determine the benefits or drawbacks of this approach. Lexicase selection is also unique in the sense that it avoids the use of summary statistics, such as averages, to compare individuals. Fitness-proportional selection, where individuals are randomly selected proportional to their fitness values, is more of a global selection strategy since it considers the whole population together but has been found to be sensitive to the specific fitness metric being used [Holland, 1992b]. Rank-based selection was adapted from fitness-proportional selection but avoids the sensitivity to the specific fitness metric since only the ranking after sorting individuals based on fitness is considered [Baker, 2014]. Elitism is a type of selection that selects just the best individuals from the whole population and copies them directly into the next generation. This is often paired with another selection method to ensure that high-performing individuals are not

lost due to random chance, which can occur for example if the best individual in a population isn't selected to compete in any tournament when using tournament selection.

Within the field of evolutionary computation, there exist four main sub-fields: evolutionary strategies [Beyer and Schwefel, 2002], evolutionary programming [Fogel, 2012], genetic algorithms [Holland, 1992a, Goldberg, 1989], and genetic programming [Koza, 1992b, Banzhaf et al., 1998].

In this dissertation, I use systems that implement genetic programming and genetic algorithms. Genetic programming is the sub-field of evolutionary computation that utilizes the genetic operators to develop computer programs that are optimized to perform a specific task [Koza, 1992b, Banzhaf et al., 1998]. In genetic programming, an individual's fitness is a measure of how well it performs a specific task. Selection is used to help direct exploration into regions within the search space around the high-fitness individuals, but selection pressure must remain balanced to allow the potential discovery of new promising regions as well. Programs in genetic programming have traditionally been represented as trees [Koza, 1992b], but other implementations such as linear genetic programming [Brameier and Banzhaf, 2007], stack-based genetic programming [Spector], and graph-based Cartesian genetic programming [Miller, 2011] have shown to be effective as well. In this dissertation, I explore tree, linear, and stack-based systems. An example of a tree is shown in Figure 3.2, which is an example of a genotype that when evaluated results in a math equation, $c/2 + x * y$. An example of a stack-based model, using an operator stack and a variable/constant stack, is shown in Figure 3.3, which when evaluated would result in the equation $1/5(2 * A + X)$.

Genetic algorithms are similar to genetic programming in how selection and variation operators are used to improve performance relative to a fitness function over many generations. Genetic algorithms differ from genetic programming in that individuals evolved in GA's are fixed size, unlike individuals in genetic programming which can adapt and change size over generations. As well, genetic algorithms are typically used to evolve a solution directly, whereas genetic programming is used to develop a program or process that can be used to solve a problem. An example structure of an individual that could be evolved with a genetic algorithm is shown in Figure 3.4. In this example, during evolution, the genetic operators could operate on the specific values in the individual to

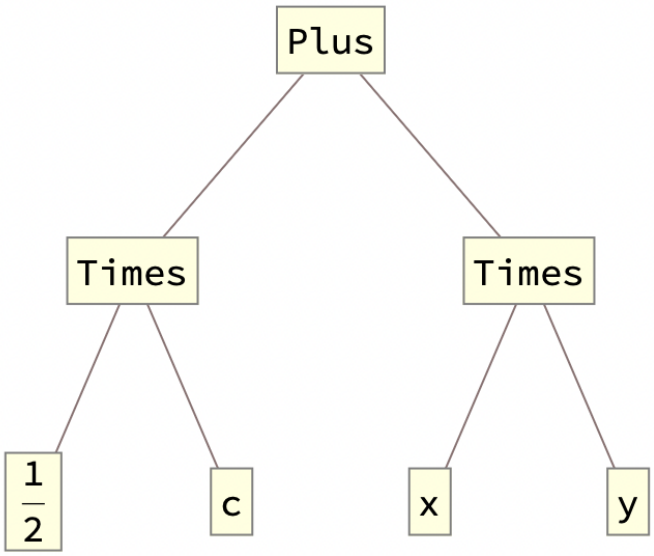


Figure 3.2 An example of a genetic programming tree is shown. When evaluated, this tree becomes the math expression $c/2 + x * y$.

attempt to find optimal values for each position in the list. Selection would be used to select individuals that have parameters that lead to better performance relative to a specific task.

3.1 Symbolic Regression

Symbolic regression is a typical application of genetic programming (GP) that develops mathematical models to fit data sets [Koza, 1992b, Banzhaf et al., 1998]. This is a form of understandable AI since the final model can be easily presented to the end user in the form of an equation. This makes it an appealing tool for researchers attempting to understand a system of study since physical laws can be extracted from the data. The equations can then be studied and used to make predictions or generate new hypotheses to help explore a system. While many different implementations exist for symbolic regression, such as DataModeler [Evolved_Analytics], Eureqa [Nutonian], AIFeynman [Udrescu and M., 2020a], etc., symbolic regression is not a solved problem. Questions remain such as how much data is needed, at what rates the genetic operators should be used, what representation is most effective, what fitness function(s) should be used, and how to balance model complexity to prevent overfitting.

Previous to this research, a set of 100 Feynman equations was used to compare the effectiveness

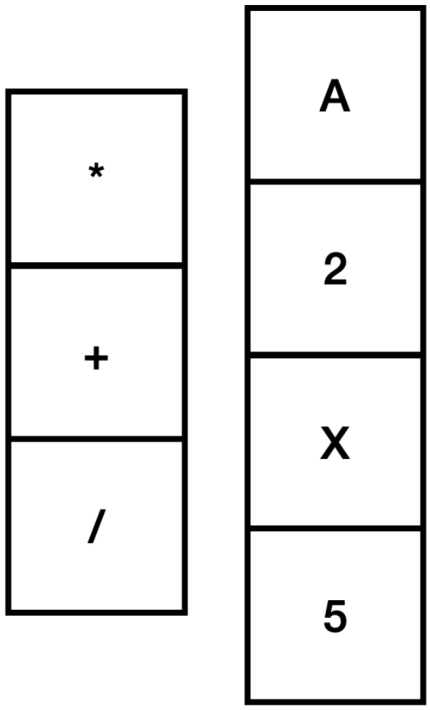


Figure 3.3 An example of a genetic programming stack-based model is shown. When evaluated, this stack becomes the math expression $1/5(2 * A + X)$.

X	1.5	0	.2	8	1	9	2

Figure 3.4 An example of an individual that could be developed by a genetic algorithm. Through evolution, the specific values in each position in this list should be optimized.

of different symbolic regression implementations [Udrescu and M., 2020a]. This benchmark data set was used to test the ability of a machine learning (ML) system to rediscover the equations using the fewest data possible. This is a useful benchmark since all of the equations are physically meaningful. Good performance on this benchmark could indicate a ML/GP system is viable for use in scientific studies attempting to discover equations describing natural phenomena. However, in this work, we do show that about a third of the problems in this set are trivial and should be excluded from future benchmark sets.

Udrescu and Tegmark themselves developed an effective ML approach, AIFeynman, that is capable of solving all 100 problems in the Feynman Symbolic Regression Database [Tegmark, Udrescu and M., 2020a]. However, on many of the more complex equations, their approach relies

heavily on dimensional analysis, translation, and neural networks that take advantage of symmetries, smoothness, and separability designed specifically to solve these types of physics problems. While their method works extremely well in solving these problems, for cases that have to rely on neural networks, large data sets are required to rediscover the equations. Also, the dimensional analysis, translations, and assumptions about symmetry and separability required significant domain expertise, rendering this a complex approach to solving general-purpose symbolic regression problems. Further, dimensional analysis requires that units both be known and recorded with the data, which may often not be the case in real-world applications. Recent work that compared many different symbolic regression methods on a broader benchmark set called, SRBench [La Cava et al., 2021], found that AIFeynman actually does not generalize well beyond the Feynman Symbolic Regression Benchmark problems. This shows that this approach is overly specialized and potentially even an example of researchers developing a system purely to perform well on a specific benchmark.

In this work, we introduce the active learning in genetic programming method (AL-GP), which uses features of population-based ML systems to determine which samples are most informative to be used in training. We apply AL-GP to 3 different population-based ML methods: StackGP, decision tree GP (DT-GP), and SEE-Segment. The goal of the AL-GP method is to create a general-purpose approach for use with population-based ML systems, such as GP, that requires no domain expertise, uses the least number of data points possible, and can guide data collection to be maximally informative. Beyond data collection for model training, the developed models could be used to design experiments to further explore systems of study. As well, the models could be used to accelerate the development process by recommending the target conditions for the system of study. An example of this could be using the developed models to design a chemical with specific target properties by recommending the conditions to produce such target properties.

Part II

Regression

Regression is a common application of machine learning where the goal is to find a model that best fits some numerical data. Symbolic regression is an application of genetic programming where mathematical expressions are evolved to fit numerical data. A successful run of symbolic regression leads to a math equation that describes the relationships that connect the input values of a dataset to the response values. Symbolic regression is an appealing ML approach for regression since the resulting model can be easily interpreted to gain insight into the relationships within the data.

This section of my dissertation focuses on how active learning can be applied to symbolic regression tasks. To apply active learning to symbolic regression I first needed a high-quality GP system for symbolic regression, so I developed StackGP. In Chapter 4, I describe StackGP, its dual implementation in Mathematica and Python, and the performance benefits realized when converted to Python. One of the features of StackGP that makes it a high-quality system for symbolic regression is its use of correlation as a fitness function as opposed to the standard RMSE fitness function. In Chapter 5, I describe the benefits of using correlation as a fitness function. From there I can move on to actually applying active learning to StackGP. In Chapter 6, I show the results of applying the active learning methods to StackGP for symbolic regression and explore how different uncertainty and diversity metrics impact the success of active learning. As well, I explore how uncertainty and diversity can be combined to form an active learning strategy that is balanced between the two.

CHAPTER 4

STACKGP

StackGP is a stack-based genetic programming system that I designed and initially created in Mathematica and have since converted into Python. It primarily draws from elements of PushGP [Spector], which is another stack-based implementation, as well as DataModeler [Evolved_Analytics], which is a tree-based implementation also built in Mathematica. StackGP is used to evolve models during the model development step of the active learning strategy. The key components of this system are described below in the following sections.

The Python version was created in response to feedback on a previous paper submission that suggested we should use open-source code instead of proprietary Mathematica functions if we want to publish in the academic community. This version is now complete and is comparable to the Mathematica version with the only significant differences being with the maximization and clustering functions since the Mathematica version used proprietary Mathematica functions not available in Python. The differences are shown in the results section.

4.1 Model Form

StackGP is the system being used for symbolic regression. Models are represented as stacks, where data types are stored in separate stacks. For symbolic regression tasks, operators are stored in the operator stack and the variables/constants are stored in another stack. The evaluation of each model is driven by the operator stack, such that it grabs the next available operator and pops off the variable/constant stack as many variables/constants as the operator needs. This continues until no more operators are left in the operator stack. An example of a model is shown in Figure 4.1, where the operator stack is shown on the left next to the variable/constant stack. The right of the arrow shows the expression that the model evaluates to.

4.2 Genetic Operators

Models are evolved using three primary genetic operators: mutation, recombination, and cloning. Two-point crossover is used as the recombination operator. Mutation has multiple choices

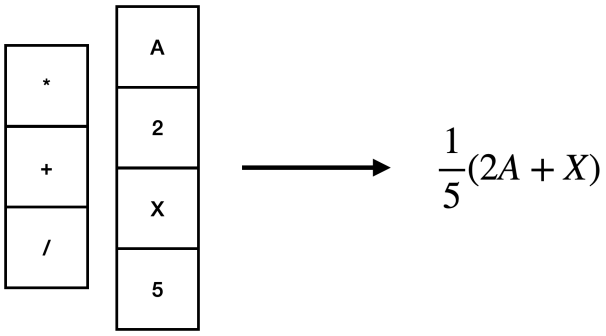


Figure 4.1 An example of a stack model and the associated equation form.

for how it will modify the parent models to produce the offspring and these choices are randomly chosen each time the operator is called. The types of mutation are as follows: variable/constant point mutation, math operator point mutation, pushing new variables and operators to the top of the stacks, trimming off the bottom of the stack, pushing new variables and constants to the bottom of the stacks, and insertion of new operators at a random position in the stack. Each mutation operator occurs with equal probability. Elitism is used to ensure that good models are not lost between generations, therefore some of the best models from each generation are cloned into the next. All of the genetic operators ensure that the new models are evaluatable. This is done by operating on both the operator and variable/constant stack synchronously.

4.3 Selection/Fitness

The models compete via Pareto tournaments, where in each tournament 5 models are randomly selected and the Pareto front of the models are returned as the winners. The Pareto front consists of all the models that are not dominated in either accuracy or simplicity. Unlike many selection strategies, multiple winners can be chosen per tournament. Correlation, specifically Pearson's R, is used as the accuracy metric, and combined stack length is used as the complexity metric. The selected models are then added to the pool of models that are assigned to be either cloned, mutated, or paired with another model from the pool of selected models for recombination.

Elitism is used to ensure the top models in each generation are not lost. Elitism is implemented by selecting Pareto front layers of the whole population until the set number of models is selected. The number of models selected via elitism is controlled by the elitism rate parameter.

Since correlation is used as the fitness function we require an alignment step to be performed at the end of an evolutionary run. This is a result of correlation considering all models of the form $a * f(x) + b$ to be equivalent for any a and b. The alignment step simply uses linear regression to find the best a and b values for each model returned. The reasons for using correlation as the fitness function are described in great detail in Chapter 5.

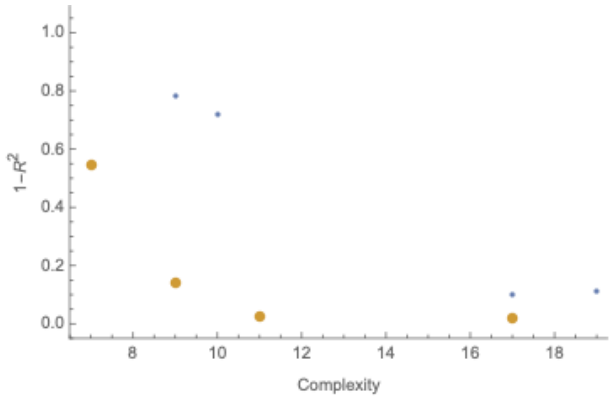


Figure 4.2 An example Pareto tournament is shown with 8 models. In this case, 4 are on the front, so the tournament would return 4 of the 8 models. These models represent the non-dominated models using the two objectives, complexity and accuracy.

4.4 Termination Criteria

Models are evolved until either a set number of generations has been completed or until a time limit has been reached. The termination criterion is set to 1000 generations with a maximum search time of 2 minutes. In the Python version, the total number of generations is generally able to be completed within the time constraint, whereas in the Mathematica version, the time limit is generally reached.

4.5 Ensemble Selection Method

A model ensemble is an essential component of the uncertainty-based active learning in this dissertation since the disagreement in an ensemble is how uncertainty is determined. To make the disagreement meaningful, the ensemble has to capture the diversity of the model population while also capturing high-quality individuals in the population. To generate a model ensemble with those qualities the first step is to cluster the training data. The models in the population of the final generation are then evaluated on each training data cluster and the best not yet selected model on

each data cluster is returned for inclusion in the ensemble. If a model has already been selected for another data cluster, the next best unselected model is returned. This ensures we don't produce a singularity where the entire ensemble consists of copies of one model. In the event that fewer than 3 data clusters are identified, rather than selecting fewer than 3 models for the ensemble, the Pareto front of the final model population is used as the model ensemble.

This method was chosen since diversity is achieved by selecting models that best fit different regions of the training space, while also ensuring high-quality models are selected since each model returned is the best or one of the best models in each region of the training space. This ensemble method was developed in some of my previous work where I developed and compared many different methods for generating ensembles for GP symbolic regression tasks [Kotanchek and Haut, 2022].

4.6 Default Evolution & Active Learning Parameters

The parameters for evolution were selected using an active learning strategy for parameter optimization. The optimization goal was to reduce the number of data points needed to find correct models on a sample problem. The parameters that were optimized were the following: mutation rate, crossover rate, spawn rate, elitism rate, tournament size, population size, selection rate, and crossover method. The training set was initialized with results from testing 3 different parameter settings on the sample problem. The active learning for parameter selection began by training models to fit those 3 parameter settings to the number of points needed to solve the sample problem. The developed models were then used to predict the best configuration and then tests were performed using the predicted parameter settings. The results of using the recommended parameter settings were then returned to the training set. This process was iterated until convergent behavior was observed. Table 4.1 shows the resulting parameter settings. The mutation rate shown in Table 4.1 refers to the rate at which a model will receive exactly one mutation.

4.7 Baseline AL Performance using Mathematica-based StackGP

An initial set of tests was run on all 100 of the Feynman Symbolic Regression problems to determine if this system and strategy are reasonable starting points for this work. In the initial

Parameter	Setting
Mutation Rate	79
Crossover Rate	11
Spawn Rate	10
Elitism Rate	10
Crossover Method	2 Pt.
Tournament Size	5
Population Size	300
Max Generations	1000
Max Runtime	2 Minutes
Selection Rate	20

Table 4.1 StackGP & Active learning Parameter Settings.

test, 37 of the 100 equations were able to be solved with just the initial random 3 data points. The minimum number of points needed by AIFeynman was 10, so StackGP outperformed AIFeynman on all of these problems. This indicates that these problems are trivially solvable and active learning is not necessary for these problems, so they give no insight into how active learning is affecting the search. Of the remaining problems, 16 were solved using fewer data points than what was reported by AIFeynman. For these problems, it seems that active learning had a positive effect on the success of the search. One of the equations needed the same number of points as AIFeynman. 18 of the problems required more data points than what was reported by AIFeynman. The 28 remaining problems were not solved within 100 iterations of active learning, so it is not possible to compare the effect that active learning had on the success of those searches. The results are summarized in Table 4.2 and full results are shown in Appendix A in Tables A.1 and A.2.

According to Udrescu and Tegmark, Eureqa is the best available commercial symbolic regression software [Udrescu and M., 2020a]. Eureqa was found to solve 71 of the 100 Feynman equations using 300 data points and 2 hours of compute time for each equation. StackGP with active learning was able to find 72 of the 100 Feynman equations, so performed similarly to Eureqa, although not all the same equations were solved. This similar performance to Eureqa indicated that StackGP was of sufficient quality to proceed with using it as the GP system for the AL experiments.

The performance of AL on each individual equation is shown in Tables A.1 and A.2 in Appendix

Performance	Total Equations
Trivial (Only 3 Points Needed)	37
Outperformed AIFeynman	16
Underperformed AIFeynman	18
Matched AIFeynman	1
Failed to Solve	28

Table 4.2 StackGP with Active Learning Performance Summary.

A. The formulae for each equation number alongside the variable ranges and sample data can be found in the Feynman Symbolic Regression Database [Tegmark] where they are ordered in the same way. The table shows the number of data points needed to solve each equation by AIFeynman, the number of data points needed to solve each equation by StackGP with active learning, the success of StackGP with active learning, and the success of Eureqa. The number in parenthesis indicates the number of repeated trials completed and averaged (median) to get the total number of points needed to solve the problem. Many of the equations were able to be tested using 100 repeated trials, although some were tested fewer times due to limited access to computing resources.

In the following, we discuss a few examples. Equation number 22 is an example of a problem that needed just 3 points to be solved.

$$\tau = rF\sin(\theta) \tag{Eq 22}$$

Looking at the equation we can see that it is relatively simple and would require only 3 operators (sin, *, *) and 3 variables (r, f, θ). It is likely easy to find, both due to its simplicity and since the terms are combined as products, which makes each variable's contribution to the response data easily distinguishable and similar in magnitude.

Equation number 3 is an example of an equation where the active learning approach with StackGP outperformed AIFeynman, needing just 42.5 points on average compared to the 1000 points needed by AIFeynman. As well, this specific equation was unsolvable by Eureqa.

$$f = \frac{e^{-\frac{1}{2}\left(\frac{\theta-\theta_1}{\sigma}\right)^2}}{\sqrt{(2\pi)}\sigma} \tag{Eq 3}$$

Performance	Total Equations
Python Outperformed	11
Similar Performance (+-2 Pts)	13
Mathematica Outperformed	12

Table 4.3 StackGP with Active Learning Version Comparison Summary.

Equation number 5 is an example of an equation that was unsolvable by StackGP with active learning and by Eureqa.

$$F = \frac{Gm_1m_2}{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \quad \text{(Eq 5)}$$

It required 1 million data points to be solved by AIFeynman. This equation is rather complicated since it has 9 variables and the contributions of each variable to the response are vastly different depending on where they are in the equation.

4.8 Baseline AL Performance using Python-based StackGP

Creating the Python version of StackGP was necessary so that the system could be fully transparent. The Mathematica version, while effective, relied on the use of proprietary Mathematica functions which were greeted negatively by reviewers early on since they could not be certain how exactly they worked. I chose to use Python since it is fully open-source and very actively used within the machine-learning community.

To compare the performance of active learning in the Python and Mathematica versions, all the equations that required more than 3 data points to solve were selected from the Feynman Symbolic Regression Benchmark. In total, there were 36 equations selected. Each problem was tested 100 times and the median number of data points needed to solve the problem are reported.

For 11 of the 36 cases, the Python version required fewer data points than the Mathematica version. In 13 of the 36 cases, the Python version and Mathematica version needed a similar number of points (+-2 points) to solve the problem. In the remaining 12 cases, the Python version did not perform as well as the Mathematica version. These results are summarized in Table 4.3. The full results are shown in the Appendix in Table A.3.

The difference between the two methods seems to lie within the different maximization functions

used to maximize the uncertainty function. Mathematica has a proprietary NMaximize function and in Python we used the Scipy Optimize Maximize function, which is open-source. Since the uncertainty function is unlikely to be convex in a typical application, a global maxima is not guaranteed to be found, so the quality of points added depends on the method used to maximize the uncertainty function. This indicates that it could be worthwhile to explore how the optimization functions and uncertainty metric can impact the success of active learning with GP.

Since the Python version was confirmed to have reasonably similar performance to the Mathematica version, and since the Python version is open-source, significantly faster, and easier to scale from a command-line interface on MSU's HPCC, we choose to use the Python version for the remainder of experiments in this dissertation.

CHAPTER 5

CORRELATION IS A BETTER FITNESS FUNCTION

Using correlation as the fitness function in StackGP for regression tasks is one of the key reasons why it performs as well as it does. This is not the standard choice of fitness function in the GP symbolic regression community though, so we explored why correlation improves performance over the standard RMSE fitness function.

We replace the traditional loss function called root mean square error (RMSE),

$$L = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{5.1}$$

where N is the number of data points i, y_i is the target output, and $\hat{y}_i$ the output calculated by the program under consideration, with the correlation function

$$R = \frac{\sum_{i=1}^{N}(y_i - \bar{y})(\hat{y}_i - \bar{\hat{y}})}{\sqrt{\sum_{i=1}^{N}(y_i - \bar{y})^2 \times \sum_{i=1}^{N}(\hat{y}_i - \bar{\hat{y}})^2}} \tag{5.2}$$

of target vs. program output.

This replacement is, however, not direct. First off, we try to maximize R^2 (or minimize $1 - R^2$), but then, in a post-processing step, we align the resulting relationship via a simple linear regression step, minimizing

$$\underset{a_0,a_1}{\operatorname{argmin}} \sum_{i=1}^{N}(|y_i - (a_1\hat{y}_i + a_0)|) \tag{5.3}$$

The essential difference between these two fitness functions is the *global* consideration the subtracted averages of equation (6.7) bring in. You can note that they are in relation to their respective output data series (target or program). They are thus entering information about the *shape* of the entire curve into the fitness function while the absolute scaling and translation are left to the linear regression post-processing step. We could say that the correlation function looks at the relative position of data points in the target dataset, and compares that to the relative position of the program/model-produced dataset.

The full results from these experiments were reported in the chapter submitted to the GPTP conference, "Correlation versus RMSE Loss Functions in Symbolic Regression Tasks" [Haut et al.,

2023a] and reprinted in this chapter with permission from Springer Nature. It was found that correlation as a fitness function when compared to RMSE is more resistant to low levels of noise, less sensitive to constants, requires fewer points to find a correct or "good-enough" solution, and is generally more successful over all degrees of dimensionality explored.

There are several reasons we discussed as being why correlation is a better fitness function. The key reason is that correlation is a more "global" fitness function since it is a measure of the shape of the function and is independent of the scaling and position of the function. This allows the number of potential candidate solutions to increase dramatically while shrinking the search space. It also allows for a solution to be selected before the correct constants are found and then linear regression is applied to find the constants, which is much faster and better suited to finding constants than GP. An example of this is shown in Figure 5.1. The figure showcases how a solution that would be considered terrible by RMSE is actually selected by the correlation fitness function and results in a near-perfect fit post-alignment.

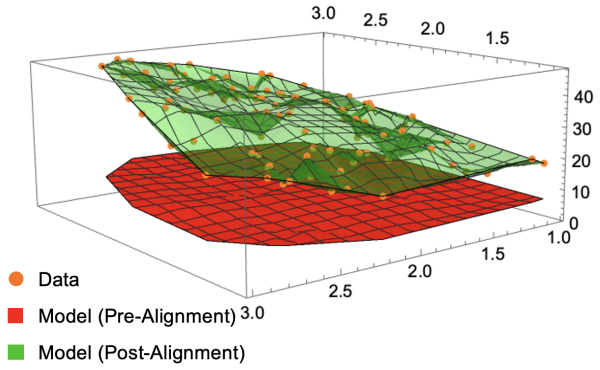


Figure 5.1 Correlation as a fitness function can identify successful models that would not be identified by RMSE. The red surface represents a model that was identified as good by the correlation fitness function and the green surface is the same model after has been aligned. Orange points represent the raw data, which consists of 100 points. Both the model and the aligned model have an R^2 value of 0.9999. The pre-aligned model has an RMSE value of 22.24 and the aligned model has an RMSE value of $2.076 * 10^{-14}$.

In this work, we compared correlation and RMSE on a set of standard symbolic regression benchmarks and explored how noise, dimensionality, and number of training points impact the performance of each fitness function.

5.1 Benchmarks

A number of different benchmarks have been proposed for symbolic regression tasks. GP started out with the symbolic regression problems used in Koza's first book [Koza, 1992a]. Over time some more complex problems were added culminating in the suggestion by White et al [White et al., 2013] to consider these as new standard benchmarks. Korns [Korns, 2013, 2014, 2015] has given a series of presentations at GPTP on systematically more difficult symbolic regression problems and their solution, increasingly relying on hybrid algorithms to solve them. Finally, Udrescu and Tegmark have proposed a collection of physical laws extracted from Feynman lectures [Udrescu and M., 2020a] as a good benchmark suite for symbolic regression algorithms.

In the following, we shall briefly discuss the former, giving examples of each, before focusing on the latter, the Feynman Symbolic Regression benchmark set.

5.1.1 Koza's Benchmarks

Koza was the first to highlight the intricacies of symbolic regression with Genetic Programming. The quartic polynomial

$$x^4 + x^3 + x^2 + x \tag{5.4}$$

was the first discussed in [Koza, 1992a]. This problem later proliferated to the following problems:

$$x^4 - x^3 + x^2 - x \tag{5.5}$$

$$x^4 + 2x^3 + 3x^2 + 4x \tag{5.6}$$

$$x^6 - 2x^4 + x^2 \tag{5.7}$$

5.1.2 New Benchmark Standards

Keijzer extended the benchmark set studied in [Keijzer, 2003] to the Keijzer instances which were further expanded by Vladislavleva et al. [Vladislavleva et al., 2008] and Nguyen et al. [Uy et al., 2011]. Typical examples are Keijzer-5:

$$\frac{30xz}{(x-10)y^2} \tag{5.8}$$

Vladislavleva-1:

$$\frac{e^{-(x_1-1)^2}}{1.2+(x_2-2.5)^2} \tag{5.9}$$

or Nguyen-5:

$$sin(x^2)cos(x)-1 \tag{5.10}$$

just to name a few.

A 2012 community survey [McDermott et al., 2012] revealed the mainly used benchmarks and was summarized and standardized in [White et al., 2013].

5.1.3 The GPTP Benchmarks

While the term GPTP benchmarks is actually broader, we focus here on the series of contributions and problem suggestions by Korns and his co-authors [Korns, 2007, 2008, Korns and Nunez, 2009, Korns, 2010, 2011, 2013, 2014, 2015].

Since this is a large set of problems, we are going to select only one here, Korns-8:

$$6.87+11\sqrt{7.23x_0x_3x_4}=6.87+29.58\sqrt{x_0x_3x_4} \tag{5.11}$$

out of 5 dimensions $x_0,...,x_4$, where some variables (x_1,x_2) do not carry information but only noise.

5.1.4 Feynman Symbolic Regression Benchmark

Here we shall mainly focus on the Feynman Symbolic Regression set of equations/data, lifted out of the lectures of Richard Feynman [Feynman et al., 1963a,b,c].

5.2 Methods

Symbolic regression was performed using StackGP. The parameters chosen for the system are shown in Table 6.1. It is important to note that the two sub-populations are evolved in parallel yet do not interact until completion. Upon completion, the two populations are merged and the fittest individuals in the combined population are then selected and returned as the final population of a run.

Parameter	Setting
Mutation Rate	79
Crossover Rate	11
Spawn Rate	10
Elitism Rate	10
Crossover Method	2-point
Tournament Size	5
Population Size	300
Independent Runs	100
Sub-populations	2
Termination Criteria	2 Minutes (wall time)

Table 5.1 Evolution Parameter Settings.

To compare the performance of using RMSE against correlation as a fitness function, we explored how noise, number of points, and dimensionality affect the resulting fitnesses of the best individuals found during evolution.

For each problem and set of conditions (noise and number of points), a total of 100 repeated independent trials were conducted and the median fitness of the best models from each trial was computed using the test data for the associated problem. To make for a simple comparison, both models trained using RMSE and correlation as their fitness functions were evaluated using RMSE on the test data. The test data consisted of 200 points generated without noise added to determine how close the evolved models are to the true generating equation.

5.2.1 Noise Introduction

Uniformly distributed multiplicative random noise was introduced to the response data $y = f(x)$ by supplying a percentage to a noise-generating function:

$$y = f(x)(1 + \epsilon), \tag{5.12}$$

where

$$\epsilon \in \left[-\frac{R}{2}, \frac{R}{2}\right] \tag{5.13}$$

is a uniformly distributed random variable from the interval $[-\frac{R}{2}, \frac{R}{2}]$.

5.2.2 Varying Number of Points

For most problems, between 3 and 183 points were used to determine how changing the number of data points affects the success of the search. This was performed by initially testing with 3 points, then adding 20 points until a total of 183 points were tested. For some of the Feynman problems, the number of points varied from 3 to 19 points incrementing by 2 to observe how small changes in the number of points impact method performance. In each independent repeated trial, the points used were generated randomly anew.

5.2.3 Dimensional Sensitivity

Sensitivity to dimensionality was explored by observing the variation in success between the different problems which vary from 1 dimension up to 9 dimensions.

5.2.4 Sensitivity to Constants

The sensitivity to identifying equations correctly when constants are introduced was explored by introducing constants of varying magnitude and determining how the error is affected by the magnitude of constants.

5.3 Results

5.3.1 The Keijzer-5 Benchmark

The results of comparing the correlation-based fitness function to the usual RMSE on the new benchmark standards with 20 to 200 points and 10% noise are shown in Figure 5.2. The results show that correlation finds more accurate models than RMSE with 10% noise for the Keijzer 5 benchmark problem.

In Figure 5.3 noise sensitivity was explored with 2,000 points with noise between 0 and 20% on Keijzer-5. While the correlation approach is sensitive to multiplicative noise and gradually deteriorates as the amplitude of noise is increased, the correlation approach generally finds more accurate models even with noise as high as 20%.

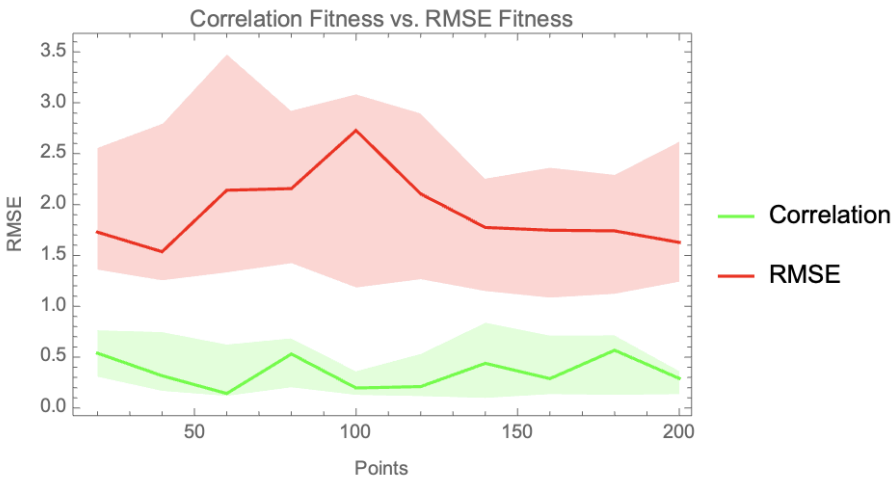


Figure 5.2 Comparing using RMSE and correlation as the fitness function on Keijzer-5 with 10% noise.

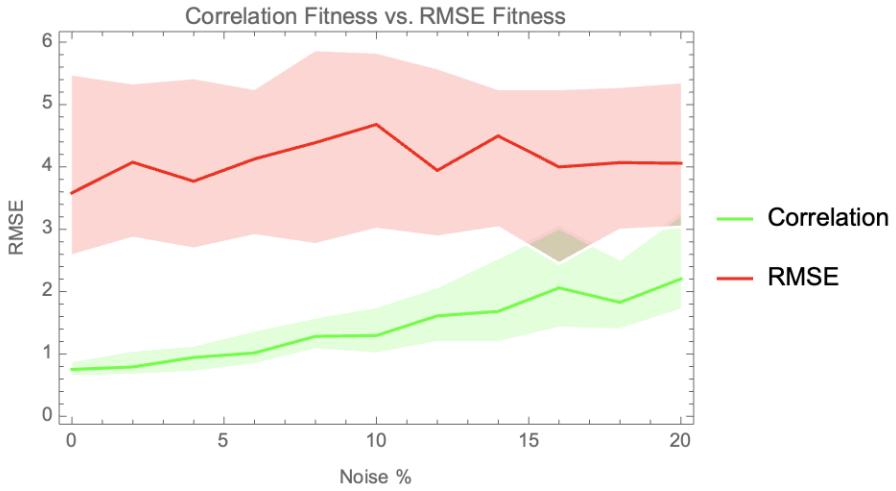


Figure 5.3 Comparing using RMSE and correlation as the fitness function on Keijzer-5 with varying noise to determine noise tolerance when using 2,000 training points.

5.3.2 The Korns-8 Benchmark

Figure 5.4 now compares the results of runs using correlation and RMSE as fitness functions to try to solve the Korns-8 benchmark problem, ranging from 3 to 183 training data points with no (0%) noise. The correlation-based fitness function consistently finds essentially perfect solutions with more than 3 points, while the RMSE-based fitness function performed relatively poorly for all numbers of data points, with only a slight improvement as the number of points increases.

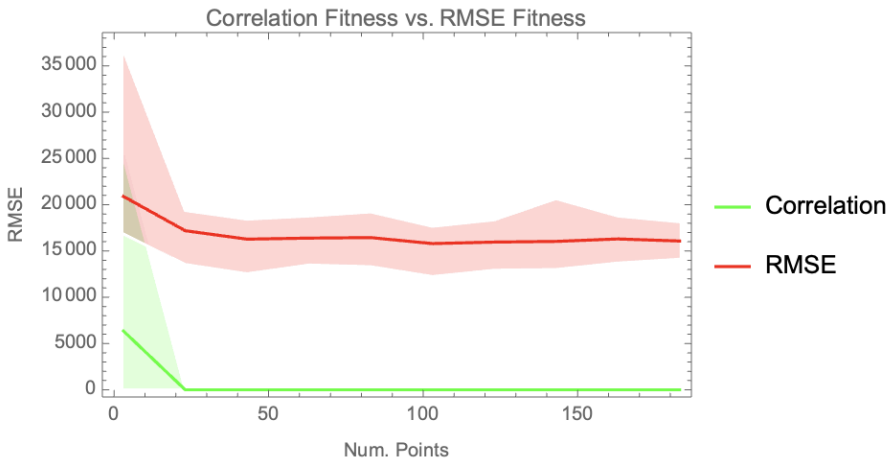


Figure 5.4 Comparing using RMSE and correlation as the fitness function on Korns-8 with 0% noise.

The noise tolerance of correlation and RMSE were also explored on Korns-8 by varying noise from 0% to 20% using 200 and 2,000 training points. The results are shown in Figures 5.5 and 5.6, respectively. With 200 data points (Fig. 5.5) the correlation approach stops outperforming RMSE when 12% or more noise is included in the data. When the data has 2,000 training points, however, as shown in Figure 5.6, we see that while the correlation approach shows sensitivity to the amount of noise present, it still finds better models with data that has 20% noise, demonstrating that more data can effectively counter noise with a correlation fitness function.

What is interesting to note is that when comparing the R^2 values between the two approaches with using 2,000 points, it can be seen in Figure 5.7 that R^2 values for all models found using correlation are 1, up to 20% noise. This indicates that the correct model form was still found in all cases and points to the alignment step as the part that is sensitive to noise. An example solution is shown in equation (5.14), compare to original function (eq. 5.11):

$$949.216 + 21.536\sqrt{x_0 x_3 x_4} \tag{5.14}$$

When looking at the R^2 values from the 200 point cases, we can see there is a very clear threshold where the signal-to-noise ratio becomes too small and the quality of models drops off significantly by using the correlation approach. The behaviour of the correlation fitness (green dots) in Fig. 5.8

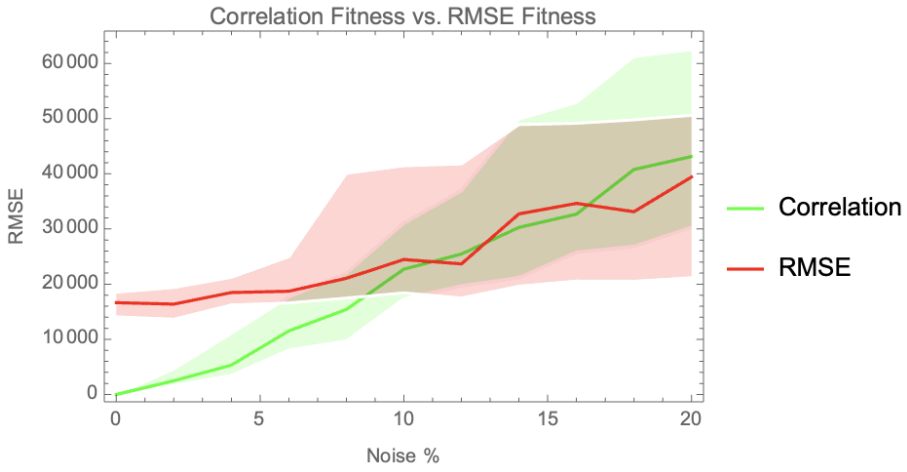


Figure 5.5 Comparing using RMSE and correlation as the fitness function on Korns-8 as noise increases from 0 to 20% with 200 training points. With 12% noise or more the correlation approach becomes comparable or worse than RMSE.

looks like a phase transition at around 10% noise. The functional relationship evolved before the transition (e.g., at 2% noise) is still correct:

$$564.468 + 23.315\sqrt{x_0 x_3 x_4} \tag{5.15}$$

However, with 14% noise (after the transition) the functional relationship is no longer correct with:

$$1406.43 + 0.133 x_0 x_3 x_4 \tag{5.16}$$

The correct variables are still found, but the square root function is now missing.

5.3.3 The Vladislavleva-1 Benchmark

Figure 5.9 compares the performance of using correlation and RMSE as fitness functions on the Vladislavleva-1 benchmark problem, ranging from 20 to 200 training data points with no noise. The correlation approach shows that it is able to consistently find models with better performance than when using RMSE as the fitness function.

Figure 5.10 explores how the two approaches vary in their sensitivity to noise. Both methods were given 200 training points and noise was varied from 0 to 20%. Correlation was observed to outperform RMSE as a fitness function until the noise level exceeded 6%. Beyond 6% the

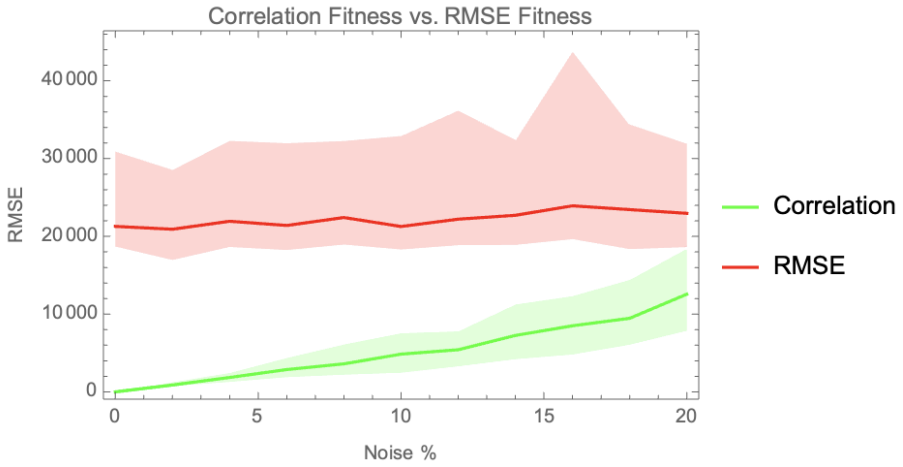


Figure 5.6 Comparing using RMSE and correlation as the fitness function on Korns-8 as noise increases from 0 to 20% with 2000 training points.

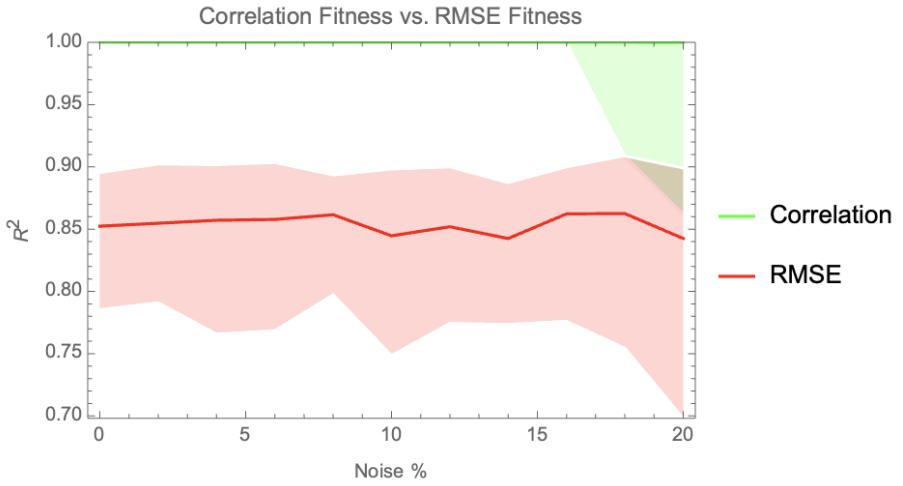


Figure 5.7 Comparing using RMSE and correlation as the fitness function on Korns-8 as noise increases from 0 to 20% with 2,000 training points using R^2 as the metric for comparison.

distributions of both methods widened significantly and the average performance of correlation as a fitness function became worse than RMSE at higher noise levels.

5.3.4 The Nguyen-5 Benchmark

The performance of correlation and RMSE as fitness functions was also compared using the Nguyen-5 benchmark problem. Figure 5.11 shows how they compare when no noise is present when training on varying numbers of points from 20 to 200. Correlation was observed to outperform RMSE as a fitness function for all of the cases between 20 and 200 points.

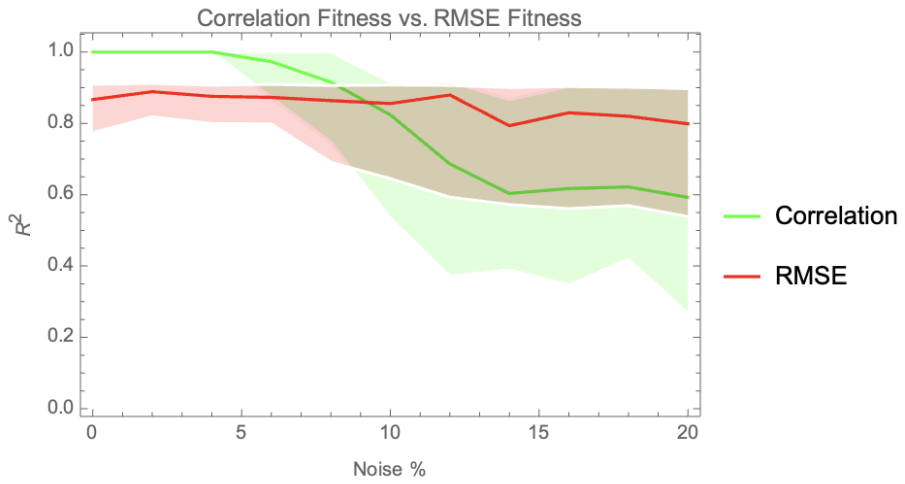


Figure 5.8 Comparing using RMSE and correlation as the fitness function on Korns-8 as noise increases from 0 to 20% with 200 training points using R^2 as the metric for comparison.

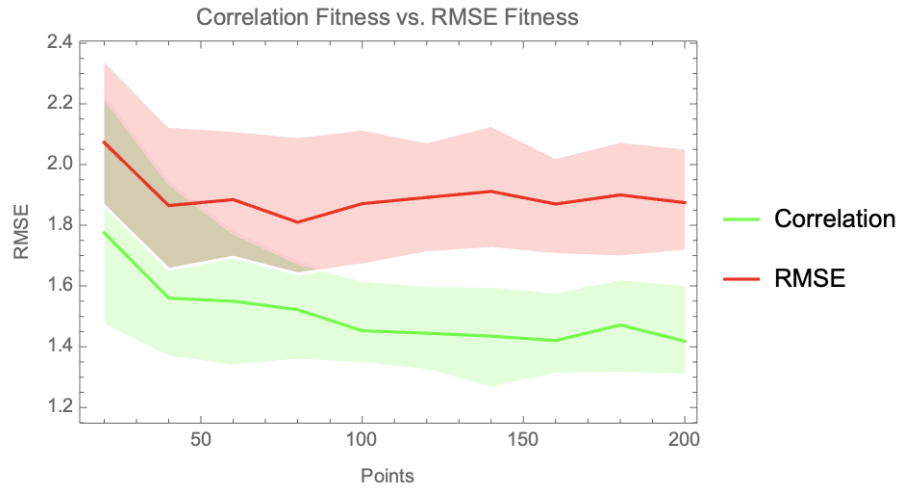


Figure 5.9 Comparing using RMSE and correlation as the fitness function on Vladislavleva-1 with 0% noise as the number of points increases from 20 to 200.

The noise tolerance of the two methods was also compared using the Nguyen-5 benchmark problem. Noise was varied from 0 to 20% with 200 training points. The results are shown in Figure 5.12. The method using correlation as a fitness function performed best until around 6% noise was present. Beyond 6% noise the two methods performed similarly with correlation having a slightly wider distribution of solutions.

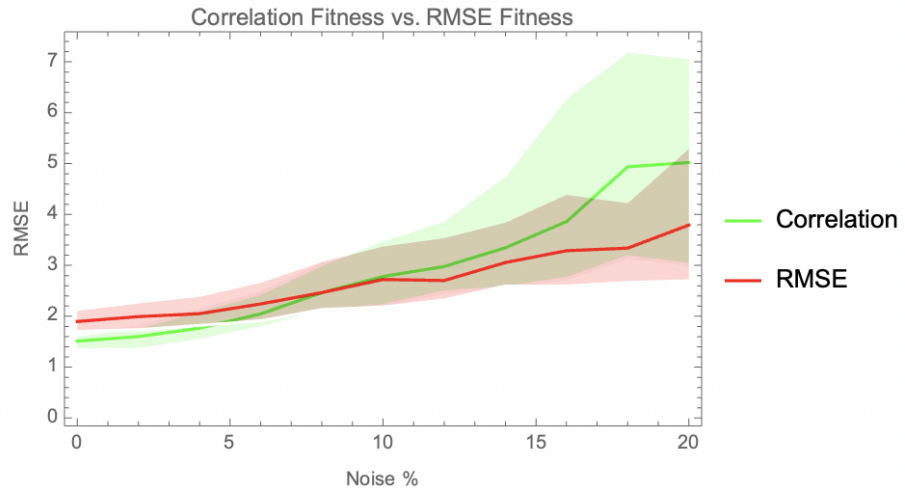


Figure 5.10 Comparing using RMSE and correlation as the fitness function on Vladislavleva-1 as noise increases from 0 to 20% with 200 training points using RMSE as the metric for comparison.

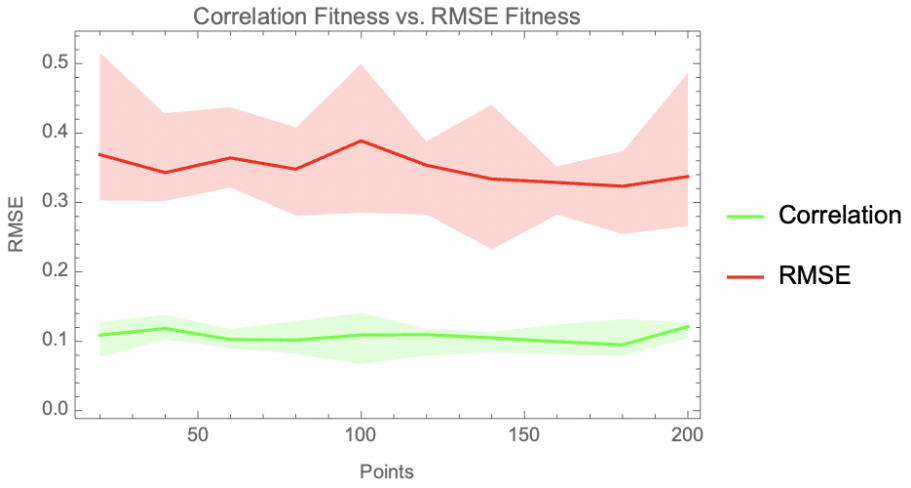


Figure 5.11 Comparing using RMSE and correlation as the fitness function on Nguyen-5 with 0% noise as the number of points varies from 20 to 200.

5.3.5 Feynman Symbolic Regression Benchmark

The results of testing the performance of the two different fitness functions on the Feynman Symbolic Regression Benchmark are summarized in Table 5.2.

With just 3 data points and no noise, the correlation approach found better models in 38 of the 100 cases and tied in performance with the RMSE approach in 11 cases. A total of 21 problems were perfectly solved with just 3 data points using the correlation approach and 10 of those 21 were not perfectly solved using the RMSE approach with just 3 data points.

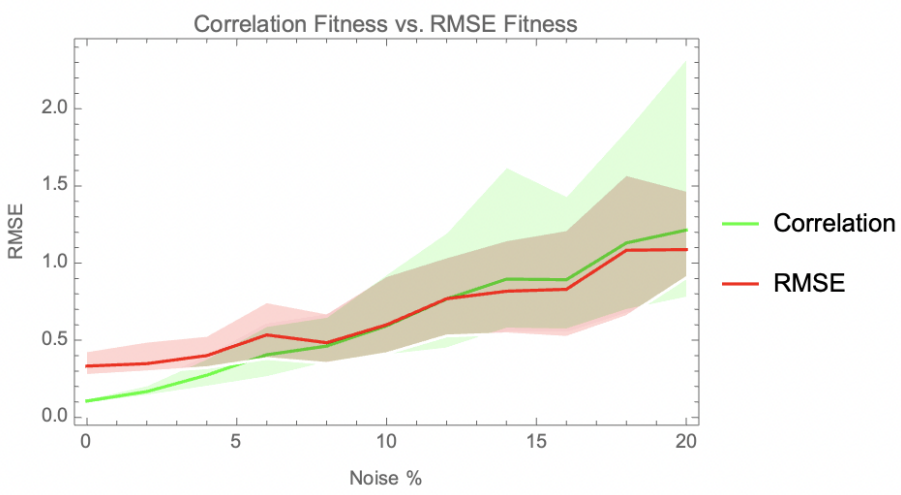


Figure 5.12 Comparing using RMSE and correlation as the fitness function on Nguyen-5 as noise increases from 0 to 20% with 200 training points using RMSE as the metric for comparison.

Number of Points	Noise %	Better	Tied	Perfectly Solved	Perfectly Solved where RMSE failed
3	0	38	11	21	10
3	10	27	0	0	0
20	0	82	17	41	24
20	10	78	0	0	0
200	0	81	17	46	29
200	10	79	0	0	0

Table 5.2 Feynman Symbolic Regression Benchmark Summary Performance Comparison of Correlation Against RMSE.

For example Figure 5.13 shows RMSE over generations on equ. (5.17) (# 8 in [Udrescu and M., 2020b]) when using correlation fitness.

$$\mu \times N_n \tag{5.17}$$

RMSE converges to 0 very quickly in a sample evolutionary run, demonstrating that this equation is trivial to solve.

As another example, we take equ. (5.18) (# 59 in [Udrescu and M., 2020b]) as one where RMSE fitness does not converge to 0, see Figure 5.14.

$$\frac{\epsilon \times E^2}{2} \tag{5.18}$$

As opposed to that, using the correlation fitness function results in runs like that depicted in Figure 5.15. As we can see from the equations, these are very simple functional relationships. We shall examine more complicated ones later, but for now will look at noise.

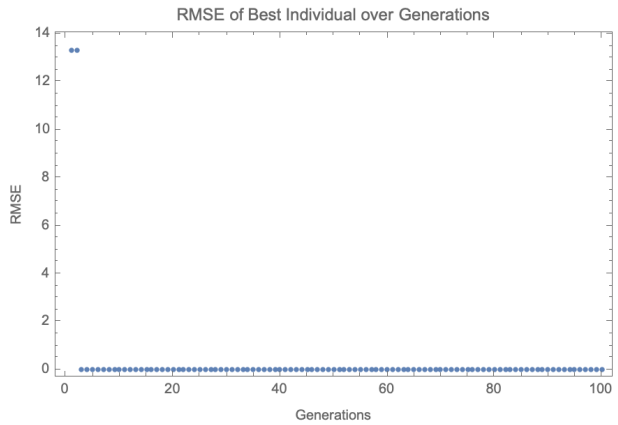


Figure 5.13 The error of best individuals over generations is shown for equ. (5.17) (# 8 in [Udrescu and M., 2020b]) when using correlation as the fitness function. We can see that equation (5.17) is trivial and is solved almost immediately.

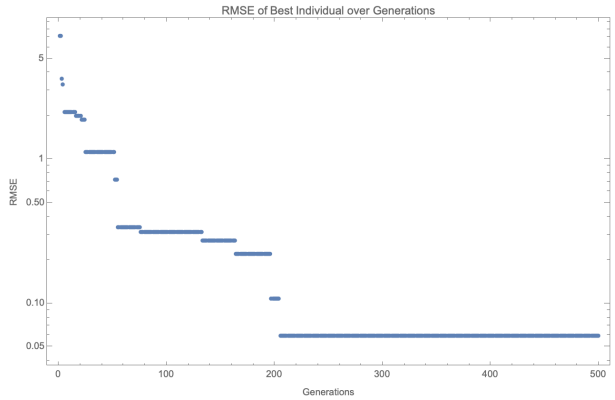


Figure 5.14 The error of best individuals over generations is shown for equ. (5.18) (# 59 in [Udrescu and M., 2020b]) when using RMSE as the fitness function. We can see that progress seems to get stuck at a local minima and stops progressing around generation 200.

With just 3 data points and 10% noise added, the correlation approach never converges to a perfect solution, but found better models in 27 of the 100 problems. On the other hand, RMSE is able to withstand the noise better, and is able to converge to perfect solutions 13 times.

Given that using three data points really is an absolute minimum, it isn't too surprising reviewers were doubtful of our results. But this seems more to be a reflection of the benchmark datasets, rather than the approach to solve it. In many cases, RMSE also solves the problem perfectly with 3 data points (see Tables B.1 and B.2 in the Appendix).

So while in trivial cases there might not be a difference between the performance of RMSE and

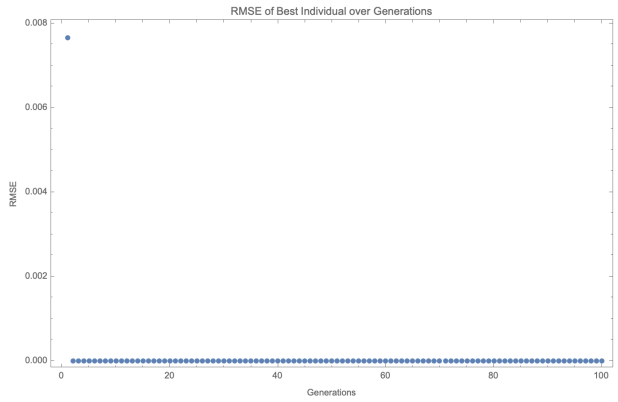


Figure 5.15 The error of best individuals over generations is shown for equ. (5.18) (# 59 in [Udrescu and M., 2020b]) when using correlation as the fitness function. We can see that this problem now becomes trivial and is solved by the second generation when using correlation instead of RMSE as the fitness function.

correlation as a fitness function, in other cases there is a difference, often a substantial one. With 20 points and no noise the correlation approach found better models in 82 of the 100 cases and was tied in another 17 cases. This means that the correlation approach beat or matched the performance of RMSE in 99 of the 100 cases of the Feynman benchmarks. As well, the correlation approach perfectly solved 41 of the problems. Of those 41, 24 were not solved using the RMSE approach.

With 10% noise added to the 20 data points, the correlation approach found better models in 78 of 100 cases. Here again, RMSE is better able to withstand the noise and converge to a perfect solution in 17 cases.

Finally, with 200 points and no noise, the correlation approach found better models than the RMSE approach in 81 of the 100 cases and was tied in 17 cases. This totals to 98 cases where the correlation approach either beat or matched the RMSE approach. A total of 46 cases were solved perfectly using the correlation approach. Of those 46 cases, 29 of them were not perfectly solved using the RMSE approach.

With 200 points and 10% noise added, the correlation approach found better models in 79 of the cases.

Figures 5.16-5.18 show three examples of noisy data and the behavior of fitness functions depending on the number of data points. These are non-trivial cases not perfectly solved by either

of the methods even with 200 points. However, we can clearly see the progress by correlation as opposed to RMSE.

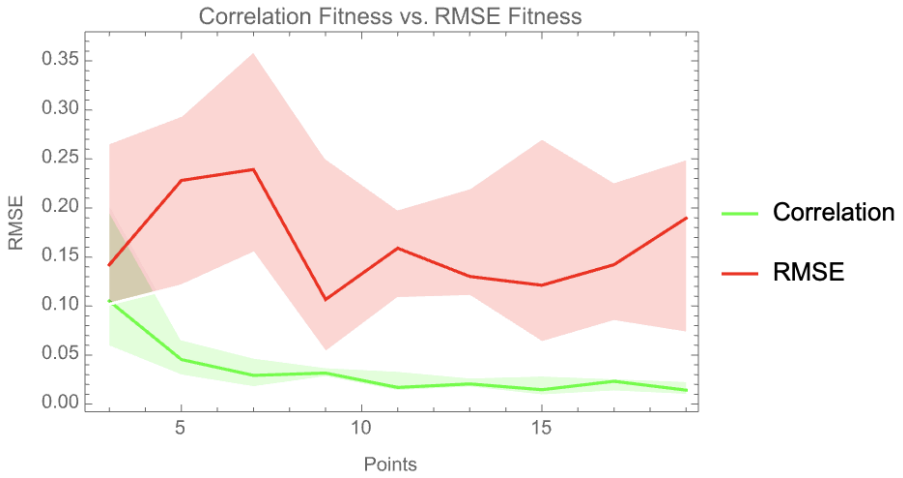


Figure 5.16 Comparing using RMSE and correlation as the fitness function on equ. (5.19) with 10% noise.

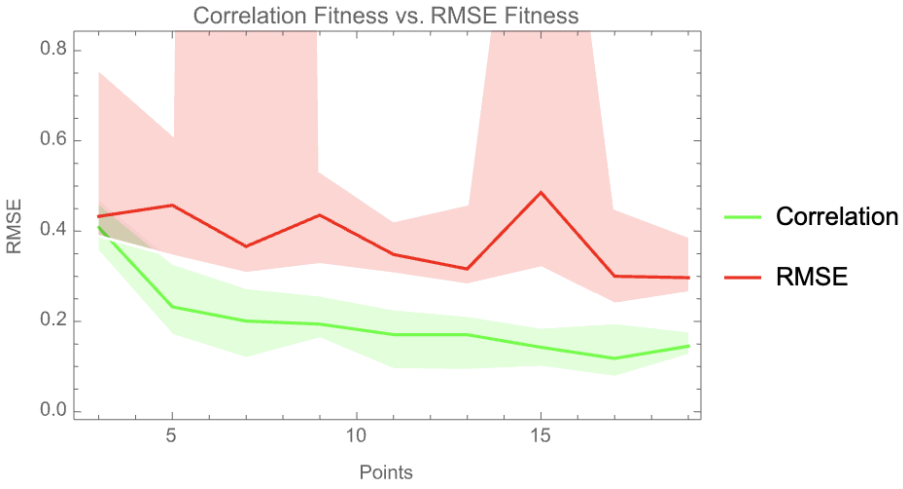


Figure 5.17 Comparing using RMSE and correlation as the fitness function on equ. (5.20) with 10% noise.

The three equations used are

$$\frac{e^{\frac{-\theta^2}{2}}}{\sqrt{2\pi}} \tag{5.19}$$

$$\frac{e^{\frac{-(\theta/\sigma)^2}{2}}}{\sqrt{2\pi}\sigma} \tag{5.20}$$

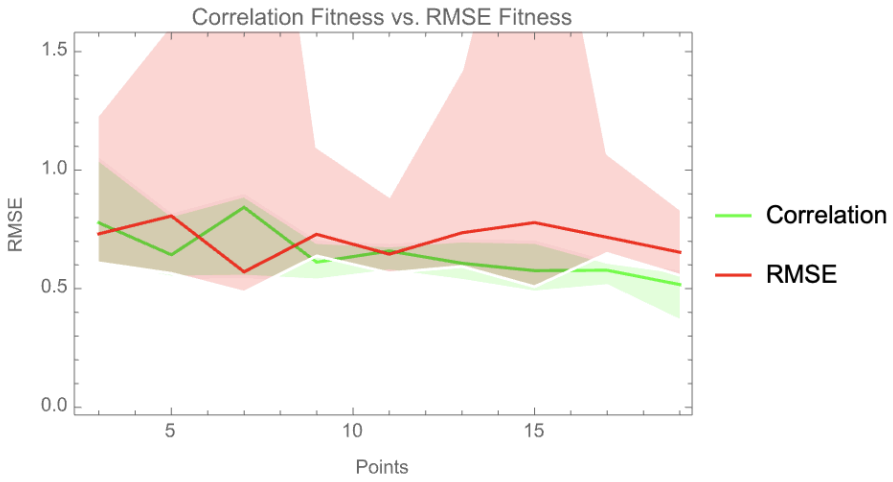


Figure 5.18 Comparing using RMSE and correlation as the fitness function on equ. (5.21) with 10% noise.

and

$$\frac{e^{\frac{-((\theta-\theta_1)/\sigma)^2}{2}}}{\sqrt{2\pi}\sigma} \tag{5.21}$$

5.3.6 Sensitivity to Constants

It was observed that equation (5.17) with 10% noise was able to be found with 0 error when using RMSE as the fitness function compared to an RMSE value of 1.35 when using correlation as the fitness function when trained with 20 points. The R^2 values from both approaches was 1, which indicates that they both found the correct general pattern in the data, but the scaling or position was off as a result of the alignment when using correlation as the fitness function. To see if the problem difficulty was altered by the introduction of a constant, equation (5.17) from the Feynman symbolic regression benchmark was modified by introducing a constant initially valued at 5 and then 50. The results are shown in Table 5.3. With no noise the problem becomes much more difficult for the RMSE fitness function approach the larger the constant, while the problem difficulty does not change relative to the constant size when using correlation as the fitness function (recall Keijzer's observation mentioned earlier). When multiplicative noise is added, the performance when using correlation does get worse as the size of the constant increases, yet it strongly outperforms when compared to using RMSE as the fitness function.

Constant	Number of Points	Noise %	RMSE	Correlation
1	20	10	0	1.35
1	20	0	0	0
5	20	10	261.69	8.74
5	20	0	276.76	0
50	20	10	3520.43	68.07
50	20	0	2151.56	0

Table 5.3 Sensitivity to Constants in (5.17). Feynman Equation # 8.

5.4 Discussion

Symbolic regression has been studied since Genetic Programming was invented more than 30 years ago. Numerous refinements have been proposed and examined, new benchmark data sets have been subjected to algorithmic variants, and results have been derived and considered in multiple forms. However, no measure has proven so efficient than simply replacing the fitness function based on absolute function values with a fitness measure that considers relative distance and leaves the absolute alignment of a functional model to a linear regression step in post-processing. Equipped with such a fitness function, many of the benchmark problems used (among them about half of the AIFeynman problems) can be safely removed from consideration as too easy. If 3 data points are sufficient to deduce the functional form of a model, then this is not a problem worthy of much attention.

For other problems, correlation did widely outperform RMSE as a fitness function and should be the function of first choice in all regression problems. It remains to be seen whether there are some other factors that allow the algorithm discussed here to shine at solving the benchmark problems considered. For example, it could be that StackGP is particularly well suited for the task and the same might not be said of other GP systems.

5.5 Conclusions

What is clear is that the fitness function in any evolutionary algorithm has an extremely important role to play as far as the performance of the algorithm is concerned. Lessons derived from the success of correlation-based fitness in symbolic regression might transfer to other tasks, like classification problems. At the very least, it could facilitate the solution of harder problems in

other domains, by training a researcher's eye on the essential functions of a system that need to be tuned to arrive at feasible solutions.

CHAPTER 6

AL-GP FOR REGRESSION

This chapter examines various methods of computing uncertainty and diversity for active learning in genetic programming. We explore how a model population in genetic programming can be exploited to select informative training data points by using a model ensemble combined with an uncertainty metric. We develop and test several uncertainty metrics and show that differential entropy performs best. We also compare several data diversity metrics and discuss the trade-off between them. Finally, we combine uncertainty and diversity using a Pareto optimization approach to allow both to be considered in a balanced way to guide the selection of informative and unique data points for training and show how that impacts the success of active learning.

6.1 Introduction

In applications of data science, the task of collecting and labelling data is often time-consuming and expensive. In some cases where data doesn't yet exist, it may be very expensive to run experiments to gather data, or possibly it could take long periods of time for experiments to complete. In these cases, it would be ideal to target specific experiments where maximal information will be gained, so fewer experiments have to be run to gain the desired insight into the system of study. In other cases, large masses of data may already exist, but the process of labelling the data is time-consuming. Here, it would be ideal to target a subset of samples, that when labelled, will provide the most information. To achieve these time-savings and cost reductions we can use machine learning (ML) not only to build models to describe these systems, but also to predict the information gained by each training sample. The process of using machine learning to iteratively select data to best inform machine learning model development is called *active learning*.

As a reminder from Chapter 2, active learning (AL) is a method used in conjunction with machine learning to actively select new training data with the goal of selecting data points that will maximally inform the machine learning model [Cohn et al., 1996].

Active learning methods have been developed and shown to be successful on a wide range of problems as I had discussed in Chapter 2, yet it has not been thoroughly explored within the field

of genetic programming.

In this contribution, we apply active learning strategies for genetic programming used in symbolic regression tasks. The goal is to exploit some of the features of GP, in particular its reliance on a population of models. More specifically, we want to utilize uncertainty and diversity measures in a model population context to accelerate the discovery of models (physics equations in our study). The idea is to look for disagreement among high-quality individuals in the population as a guide to locate informative data points to add to the training set.

6.2 Related Works

Active learning methods for machine learning have shown to be very successful in applied settings to improve the method of labelling and collecting data with various machine learning types. AL has recently been demonstrated to significantly reduce the labelling efforts required for labelling data associated with identifying heart disease [El-Hasnony et al., 2022]. The authors demonstrated that they could find more accurate models using fewer data points when compared to a random point selection strategy.

AL has been applied to genetic programming classification tasks as well. Using an ensemble of GP models, the models "vote" on the class of data pairs, and points are only labelled when the committee of developing models encounters pairs that can't be classified [De Freitas et al., 2010]. This was found to reduce the total effort needed to label training points, since only a subset had to be labelled before finding accurate models. Where GP training sets are large, AL has been successfully applied by selecting sub-samples to be used for training [Lasarczyk et al., 2004, Curry et al., 2007]. In [Curry et al., 2007] AL is performed by segmenting the data into smaller blocks and training the models using one randomly selected block at a time using uniform probability. As training continues, bias is introduced into the probability by increasing the tendency to select blocks that haven't been seen in a while, as well as blocks where the models performed poorly during training. AL for sub-sampling with genetic programming was found to decrease training times to find better binary classification models by an order of magnitude [Curry et al., 2007]. In [Lasarczyk et al., 2004] subsets were selected by dynamically developing a fitness case topology

that could be used to create minimally related subsets of data. In this context, the strength of a relationship between two training cases was indicated by the number of individuals that were able to solve both training cases.

In the discovery of biological networks AL methods have also been employed successfully [Sverchkov and Craven, 2017]. Several different approaches were explored by the authors for determining which new data points would be maximally informative for a wide range of machine learning models, including Boolean networks, causal Bayesian networks, differential equation models, etc. One approach the authors explored was the maximum difference method in which two best-fit models are chosen and a new data point is selected where those two best-fit models have the largest difference in predictions. They also examined entropy score maximization. In that method, a new data point is selected that maximizes an entropy score, where entropy can be thought of as the amount of information to be gained by gathering that data point. The entropy score H_e is computed as follows:

$$H_e = -\sum_{x=1}^{x_e} \frac{e_x}{|M|} \log_2 \frac{e_x}{|M|}$$

where M is the set of Boolean networks, x_e is the number of network states for a given data point, and e is the set of all potential data points.

In chemical engineering AL has been applied to expedite a reaction screening process by only selecting a subset of maximally informative experiments to complete rather than by exhaustively performing all possible experiments [Eyke et al., 2020]. This was done by training neural networks and using them to select a subset of experiments that maximized the information gain. Maximal information gain was determined by looking at the standard deviation of an ensemble of neural networks.

Kotanchek et al.(2009) used genetic programming for active design of experiments, where models developed by a GP system are used to find optimal conditions in a system of study. Active design of experiments is an application of active learning, where it has the goal of designing experiments that have specific properties or yield maximal information. The authors proposed to employ ensembles of models from symbolic regression to find regions of uncertainty in order to

gather new data with high information content. While this method has been proposed for how an active learning method using model ensembles could be applied to GP for symbolic regression, there has yet to be any research showing how active learning methods affect the performance of GP symbolic regression tasks or how the method to quantify uncertainty affects the quality of points selected for inclusion in the training data. Also, it is yet to be shown that this idea of selecting an ensemble from a model population and searching for points of high uncertainty or disagreement among models is generalizable to any machine learning method where a population of models is available.

6.3 Methods

We compare two classes of active learning: uncertainty and diversity-based. The implementations are described in detail below. We use two random sampling methods as a baseline to compare the performance of the active learning methods. The key features of the GP system we used, StackGP, are also discussed.

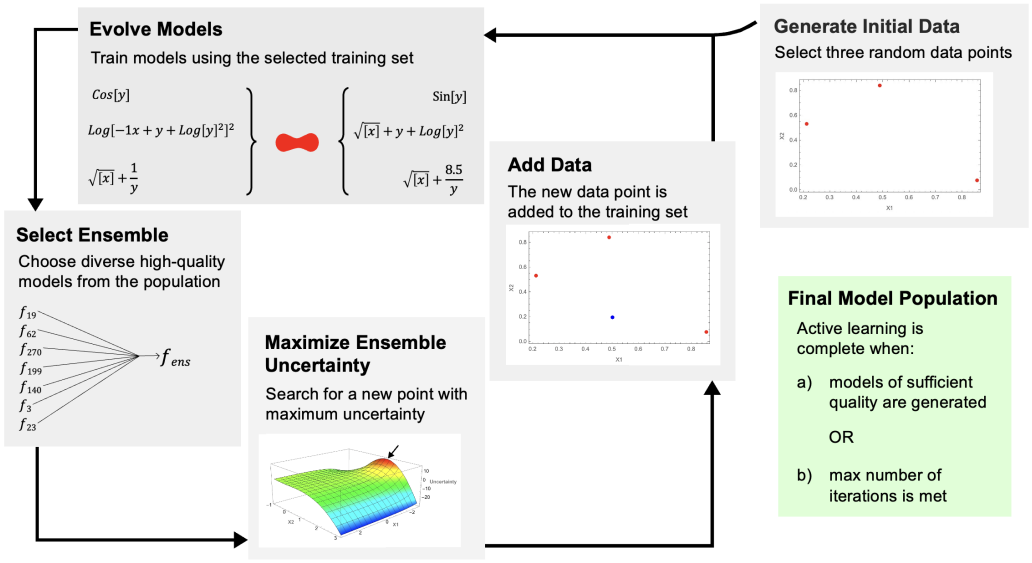


Figure 6.1 An overview of the iterative active learning approach. It begins with an initially randomly selected dataset. It then iteratively evolves models and selects new training points that maximize uncertainty of an ensemble of models. By maximizing ensemble uncertainty to select new training samples, points with relatively high information content are added to the training set each iteration.

6.3.1 Active Learning

Two general types of active learning were implemented to work with StackGP for the purpose of accelerating the development of models to fit physics data from the Feynman Symbolic Regression Dataset (Tegmark). The first type of active learning explored was uncertainty-based, a model-driven approach to active learning, where an ensemble of diverse, high-quality models from a population was used to search for regions in the search space where there was high uncertainty or disagreement between the models. The second type of active learning explored was diversity-based active learning, where new points are selected that differ maximally from the points already in the training sample. This second type of active learning is a data-driven approach rather than a model-driven approach. The first type of active learning is summarized in Figure 6.1.

Both types of active learning methods were implemented to determine how they each impact the success of evolution in genetic programming symbolic regression tasks. Several different uncertainty and diversity metrics are implemented to determine their respective impact on the success of the task. Success of active learning by maximizing uncertainty would indicate that the diversity of the population can be utilized to guide the collection of informative data. Success of diversity sampling would indicate that GP symbolic regression model development benefits from improved data sampling.

6.3.1.1 Maximizing Uncertainty

Several different uncertainty metrics were explored to determine how different measures impact the success of active learning, and which approach would generally work best. As an overview, each approach begins by selecting an ensemble of models using the same method, then a function that uses the specific uncertainty metric along with the ensemble and current training set is created. This function is then fed to an optimizer to search for regions of relatively high uncertainty. The most uncertain point found is then returned and selected to be added to the training set. In total, there were 6 different uncertainty maximization approaches tested which varied in how they quantified disagreement, whether outlier predictions were considered, and which optimizer was used. The steps and methods will be described in greater detail below and the entire process is depicted in

Algorithm 6.3.

Generating the ensemble is the first step in uncertainty-based active learning. The goals for generating the ensemble were to capture diverse, high-quality individuals from the population while keeping the size of the ensemble relatively small so that the computational cost of optimizing uncertainty is reasonable. The diversity goal is essential to the success of active learning since disagreement between models is a necessary requirement. The method chosen to capture both diversity and quality from the model population works by clustering the training data using the input space and selecting a model that best fits each cluster, ensuring no model is selected more than once. If a model is already selected by another cluster, the next best unselected model is chosen. The minimum number of clusters is set to 3 and the maximum is set to 10. Thus, 3-10 models are chosen for inclusion in an ensemble. Data clustering was chosen with the intent to capture diversity by focusing on models that have biases for different regions of the training space. Quality in the population would be captured since only models with the best fitness were selected for each cluster. The algorithm to generate the ensemble is described in detail in Algorithm 6.1.

```
procedure EnsembleSelect(models,trainingData,responseData)
    selectedModels ← []                                          ▷ Initialize ensemble
    nClusters ← min(len(trainingData), 10)                       ▷ Determine number of clusters
    clusters ← KMeans(nClusters).fit_predict(trainingData)
    for i = 0;i++; i < nClusters do                              ▷ Loop over data clusters
        modelErrors ← computeError(models, clusters[i])
        sortedModels ← sortBy(models, modelErrors)
        j = 0
        while sortedModels[j] in selectedModels do               ▷ Find best unselected model
            j++
        end while
        selectedModels = join(selectedModels, sortedModels[j]    ▷ Add to ensemble
    end for
    return selectedModels                                        ▷ Return ensemble
end procedure
```

Algorithm 6.1 Ensemble generation process to select diverse high-quality models.

The second step of this method is to utilize the specified uncertainty function with both the current training data and the selected ensemble. The function is then given to the optimizer with the search space boundaries to find a point of relatively high uncertainty. In the case that an already selected point is re-selected, a new search is initiated within a random sub-region until a unique point is added. This ensures that new information is added in each iteration to the training set.

The two methods used for optimization were Scipy Optimize's minimize and differential evolution (SciPy, SciPy).

In total 5 different uncertainty metrics were used, shown by Equations 6.1 to 6.5, where Equation 6.5 is used twice, once with Scipy's minimize function for optimization, and a second time with Scipy's differential evolution function for optimization. Scipy's differential evolution using the default 'best1bin' strategy is shown in pseudocode in Algorithm 6.2.

$$\Delta = \frac{\text{Std(EnsembleResponses)}}{\text{Mean(Abs(EnsembleResponses))}} \tag{6.1}$$

$$\Delta = \frac{\text{TrimmedStd(EnsembleResponses, 0.3)}}{\text{TrimmedMean(Abs(EnsembleResponses), 0.3)}} \tag{6.2}$$

$$\Delta = \frac{\text{Std(EnsembleResponses)}}{\text{TrimmedMean(Abs(EnsembleResponses), 0.3)}} \tag{6.3}$$

$$\Delta = \text{Std(EnsembleResponses)} \tag{6.4}$$

$$\Delta = \text{DifferentialEntropy(EnsembleResponses)} \tag{6.5}$$

$Pop \leftarrow$ N $RandomVectors$ ▷ Generate initial random population
$b0 \leftarrow BestVector(Pop, ObjectiveFunc)$ ▷ Get best initial vector
while $iter < maxIter$ **do** ▷ While less than max iterations
  for $i = 0; i < len(pop); i++$ **do** ▷ iterate over each population member
    $new \leftarrow b0 + mutation * (pop[rand1] - pop[rand2])$. ▷ Generate new vector
    for $j = 0; j < len(new); j++$ **do** ▷ Iterate over each value in individual vector
      if $rand() < recombinationRate$ **then** ▷ If random value less than recombination rate
        $pop[i][j] = new[j]$ ▷ Replace value in vector with value from new vector
      end if
    end for
    if $Fitness(new, ObjectiveFunc) < Fitness(b0, ObjectiveFunc)$ **then** ▷ If new vector is the best
      $b0 \leftarrow new$ ▷ Replace best with new vector
    end if
  end for
end while
Return $b0$ ▷ Return best vector

Algorithm 6.2 Differential Evolution.

$TrainingData \leftarrow 3StartingPoints$ ▷ Generate initial random training data
$Models \leftarrow RandomModels$ ▷ Generate initial random models
$Models \leftarrow Evolve(TrainingData, Models)$ ▷ Train models on starting data
while $BestModelError \neq 0$ **do** ▷ While perfect model not found
  $Ensemble \leftarrow EnsembleSelect(Models).$ ▷ Select ensemble of models
  $NewPoint \leftarrow MaxUncertainty(Ensemble)$ ▷ Find point of max uncertainty
  if $NewPoint \subset TrainingData$ **then** ▷ If point already selected
    $NewPoint \leftarrow MaxUncertainty(SubSpace(Ensemble))$ ▷ Search a subspace
  end if
  $TrainingData \leftarrow Append(TrainingData, NewPoint)$ ▷ Add new point
  $Models \leftarrow Evolve(TrainingData, Models)$ ▷ Evolve new models with new data using best models to seed evolution
end while

Algorithm 6.3 Active Learning Process Using Uncertainty.

6.3.1.2 Point Diversity

A data-driven active learning approach was also explored, aiming to maximize data diversity rather than maximize ensemble uncertainty. The goal was to determine if GP evolution for symbolic regression tasks would benefit significantly from improved sampling of the data for training. Two different metrics were used to quantify diversity: point distance and point correlation. Point distance was implemented by measuring both the minimum and average Euclidean distance to all points in the training set. Point correlation was defined as the average correlation to all points in the training set. When selecting a new point, the goal was to either maximize the distance or minimize the correlation to the current training set.

To minimize the correlation when selecting a new point, Pearson's R^2 was computed between each point and the potential new point. The equation for computing Pearson's R is shown in Equation 6.6. Here y represents the new training point, $\hat{y}$ represents a point already in the set, and each instance i represents the value in the ith dimension of the point. The overall method for computing the joint correlation of a new point to the training set is summarized in Algorithm 6.4.

$$R = \frac{\sum_{i=1}^{N}(y_i - \bar{y})(\hat{y}_i - \bar{\hat{y}})}{\sqrt{\sum_{i=1}^{N}(y_i - \bar{y})^2 \times \sum_{i=1}^{N}(\hat{y}_i - \bar{\hat{y}})^2}} \tag{6.6}$$

1: **procedure** JOINTCORRELATION($trainingSet$,$newPoint$)
2: $r2Values \leftarrow [PearsonR(trainPt, newPoint)^2$ for $trainPt$ in $trainingSet]$ ▷ R^2 vals
3: $avgCorr \leftarrow mean(r2Values)$ ▷ Compute average correlation
4: Return $avgCorr$
5: **end procedure**

Algorithm 6.4 Joint Correlation.

```
TrainingData ← 3StartingPoints                          ▷ Generate initial random training data
Models ← RandomModels                                   ▷ Generate initial random models
Models ← Evolve(TrainingData, Models)                   ▷ Train models on starting data
while BestModelError ≠ 0 do                             ▷ While perfect model not found
    NewPoint ← MaxDiversity(TrainingData)               ▷ Find point of max uncertainty
    if NewPoint ⊂ TrainingData then                     ▷ If point already selected
        NewPoint ← MaxUncertainty(SubSpace(TrainingData))   ▷ Search a subspace
    end if
    TrainingData ← Append(TrainingData, NewPoint)       ▷ Add new point
    Models ← Evolve(TrainingData, Models)   ▷ Evolve new models with new data using best models to seed evolution
end while
```

Algorithm 6.5 Active Learning Process Using Diversity.

6.3.1.3 Benchmark Testing

Each active learning approach was compared on a benchmark set of 35 of the 100 equations from the Feynman Symbolic Regression Dataset [Udrescu and M., 2020a]. As previously discussed in Chapters 4 and 5, these particular 35 problems were selected since they were thought to be most appropriate for a study in active learning.

6.3.2 StackGP

StackGP is a stack-based genetic programming implementation in Python [Haut et al., 2022] and is available here: https://github.com/hoolagans/StackGP.

6.3.2.1 Model Structure

Similar to PushGP (Spector), StackGP models use multiple stacks, where the model evaluation is driven by an operator stack while variables, constants, and other data types are stored on separate stacks. For symbolic regression tasks, we have a total of 2 stacks, the operator stack and the variables/constants stack.

6.3.2.2 Correlation Fitness Function

Unlike many symbolic regression implementations that use (R)MSE as the fitness function, we employ correlation as the fitness function, together with a linear scaling post-processing step. The reasons for using this fitness function were covered here in Chapter 5. The fitness is optimized during search by first maximizing R^2, which is computed using Equation 6.7, where N is the

number of data points i, y_i is the target output, and $\hat{y}_i$ the output calculated by the model.

$$R = \frac{\sum_{i=1}^{N}(y_i - \bar{y})(\hat{y}_i - \bar{\hat{y}})}{\sqrt{\sum_{i=1}^{N}(y_i - \bar{y})^2 \times \sum_{i=1}^{N}(\hat{y}_i - \bar{\hat{y}})^2}} \tag{6.7}$$

The search is then completed using a post-processing step, which aligns the resulting models via a simple linear regression step (eq. 6.8), minimizing

$$\underset{a_0,a_1}{\operatorname{argmin}} \sum_{i=1}^{N}(|y_i - (a_1\hat{y}_i + a_0)|) \tag{6.8}$$

6.3.2.3 Algorithm

An overview of the algorithm is shown in Algorithm 6.6. The parameters used to run the algorithm are shown in Table 6.1. Note that crossover and mutation calls in the algorithm are simplified and actually represent applying crossover and mutation to the correct fractions of models as shown in the parameters.

Crossover is performed using a 2-point crossover operator where two points are selected in the operator stack of each parent and the operators, along with the associated variables and constants between the points, are swapped between the parents. Mutation has several different forms, each occurring with equal probability: random replacement of a variable, random replacement of an operator, pushing a random operator to the top of the operator stack and pushing variables/constants to the second stack when arity is greater than 1, popping a random number of operators off the operator stack and the correct number of variables/constants off the second stack, inserting a single operator at a random position in the stack, 2-point crossover with a random model, and appending a random operator to the bottom of the operator stack. There is then a repair mechanism that will push variables and constants to the top of the second stack if - after mutation - there are not enough items in the variable/constant stack for the operators.

The tournament selection method used was Pareto tournament selection, where correlation and complexity were the two objectives. Complexity was measured as the combined stack lengths.

Parameter	Setting
Mutation Rate	79
Crossover Rate	11
Spawn Rate	10
Elitism Rate	10
Crossover Method	2 Pt.
Tournament Size	5
Population Size	300
Selection Rate	20
Parallel Runs	4
Generations	1000

Table 6.1 StackGP & Active learning Parameter Settings.

procedure EVOLVE($trainingData$,$models$)
  for generations 1 to 100 **do**
    $models \leftarrow setModelQuality(models, trainingData)$
    $newPop \leftarrow ElitismSelection(models, 20\%)$
    $models \leftarrow tournamentSelection(models)$
    $newPop \leftarrow newPop + crossover(models) + mutation(models)$
    $newPop \leftarrow newPop + randomNewModels$
    $newPop \leftarrow deleteDuplicates(newPop)$
    $models \leftarrow newPop$
  end for
  $alignedModels \leftarrow alignment(models, trainingData)$
  Return $alignedModels$
end procedure

Algorithm 6.6 StackGP Search Algorithm.

6.3.3 Random Sampling

As a baseline, we used random sampling of data points from uniform and normal distributions to determine if an active learning method improves learning progress over a naive sampling of training data. Uniform random sampling was chosen since it is a commonly used distribution and would likely be a first choice for naively sampling data. A normal distribution was selected since according to the central limit theorem, normal distributions tend to arise in nature, so a data set sampled from natural processes would likely be a normal distribution.

To create a fair comparison against the active learning methods, a simple substitution was made where instead of using active learning to maximize uncertainty or diversity, a random point was

added in each iteration. Beyond that substitution, the algorithm remains the same.

The normal distribution for each variable was defined using the midpoint between the sampling bounds as the mean and $1/6$ of the difference between the upper and lower bounds as the standard deviation. This places 99.8% of the distribution between the upper and lower bounds of each variable. If a point is sampled beyond a boundary it is adjusted to be on the boundary instead, although this is unlikely to occur frequently.

6.4 Results and Discussion

Several different approaches for computing uncertainty and diversity were compared using the Feynman Symbolic Regression Dataset. We then combine diversity and uncertainty using a Pareto optimization approach and compare that multi-objective method to using both uncertainty and diversity alone. The Pareto approach is then tested on two additional benchmark problems from the SRBench benchmark set.

6.4.1 Active Learning Uncertainty Sampling

The results of comparing the different uncertainty-based active learning methods are shown in Figures 6.2 and 6.3 and the full table is in Appendix C as Table **??**. Figure 6.2 uses uniform random sampling as the baseline for comparison, shown as the blue line in the figure. We also include normally distributed random sampling for comparison as the red distribution. The results show that the relative uncertainty measures, where we divide by the mean or trimmed mean, do not consistently perform better than uniform random sampling. The non-relative uncertainty measure performed well more consistently with the methods that use differential entropy performing best. The fact that standard deviation alone as an uncertainty metric performs consistently well is appealing since it is very cheap and easy to implement relative to some of the others. Differential entropy when using differential evolution as the optimizer performed best. The fact that differential evolution as the optimizer worked best with differential entropy likely indicates that the surface is highly non-convex, so differential evolution was better able to search the uncertainty space.

Figure 6.3 compares the performance of each method against uniform random sampling for each problem and displays the number of times each method outperforms or underperforms random

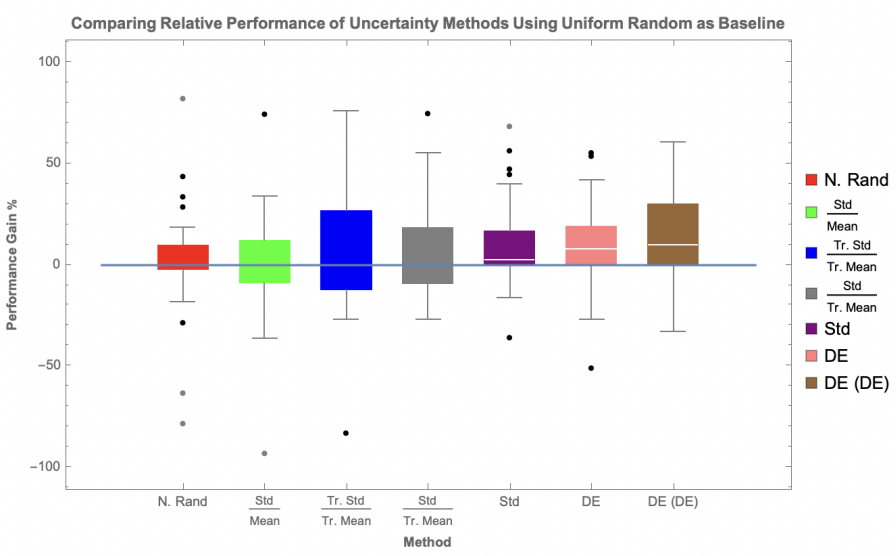


Figure 6.2 Comparing Relative Performance of Uncertainty Methods Using Uniform Random Selection as Baseline. Shown here are the performance differences of AL uncertainty methods compared to uniform random selection as the baseline (blue line) and normally distributed random selection (red distribution). We see that using the relative uncertainty measures where we divided by the mean we get inconsistent performance, sometimes performing much better than random but sometimes performing much worse. The non-relative approaches all consistently perform better than random selection with the methods that use differential entropy performing best. Using differential entropy with differential evolution (brown) we observe the best performance. The distributions represent the median performances of 100 independent runs across all test problems. For completeness, there is one point not shown for the std/tr. mean approach that is around -200.

sampling. If a method outperforms random sampling that means that the method required fewer points to solve a problem. If a method underperforms random sampling that means that the method required more points to solve a problem. The results show that the methods using differential entropy work best, outperforming in the most number of cases and underperforming in the fewest number of cases. The differential entropy method that used differential evolution as the optimizer worked better than just using differential entropy with SciPy Optimize's minimize function. This indicates that differential evolution was able to search the uncertainty surface more effectively. The results also show that the relative uncertainty methods that divided the mean or trimmed mean were not consistent in their performance, frequently having a similar number of cases where the methods outperformed and underperformed.

We see that the relative measures sometimes perform well and sometimes perform poorly, but

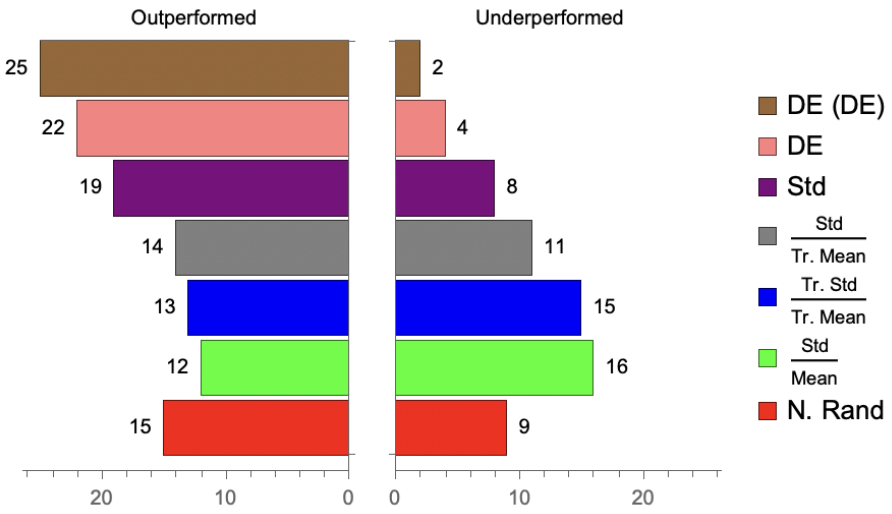


Figure 6.3 Comparing Performance of Uncertainty Methods Against Uniform Random Selection. Each method is compared to uniform random sampling and the number of times that the method outperforms and underperforms is reported. The number of times each method outperforms is shown on the left and the number of times each method underperforms is shown on the right. Outperforming means that a method used fewer points than uniform random sampling. Underperforming means that it required more points. Ties are not counted but can be easily determined by taking the difference of 35 and the two values reported. The results show that the methods that use differential entropy work well most consistently, outperforming more frequently and underperforming infrequently. We can also see that the relative uncertainty measures were very inconsistent in their performance.

on average they are centered around the baseline performance. The original assumption was that the relative uncertainty measures would be appealing since it was thought that they would reduce a bias towards selecting points where the predicted response is larger and thus naturally leads to wider distributions of the ensemble. This may have been the case occasionally where those methods did perform much better than uniform random sampling, but they were not consistent. Looking at their formulations there is a risk of selecting points where the mean is near 0 which results in asymptotic behavior of the uncertainty function.

Considering the results, we also see that of the two random sampling methods, normally distributed random sampling seems to perform a bit better than uniform sampling. This indicates that if a researcher does not want to use active learning to guide their data collection, they would typically be better off using a normal distribution than a uniform distribution for their samples.

6.4.2 Active Learning Diversity Sampling

The different metrics for determining point diversity were compared to determine if there are clear differences in what they are measuring and also to ensure there aren't any obvious flaws with any of the metrics. When comparing minimum distance and average distance an initial randomly generated training set with 3 data points in 3 dimensions was generated. Figure 6.4 shows the comparison where new points were selected iteratively to add to the training set using the minimum distance metric for selection. We can see that the correlation, R^2 is actually pretty weak between the two, indicating they are providing different measures. As well, we recorded the Spearman Rho, rank-correlation, since that indicates if the methods are ranking points similarly or not. If methods rank points similarly, then they would likely not provide unique information if used as a diversity metric. It was found that the Spearman Rho was 0.44, which means that the two methods are ranking points differently and could provide unique information.

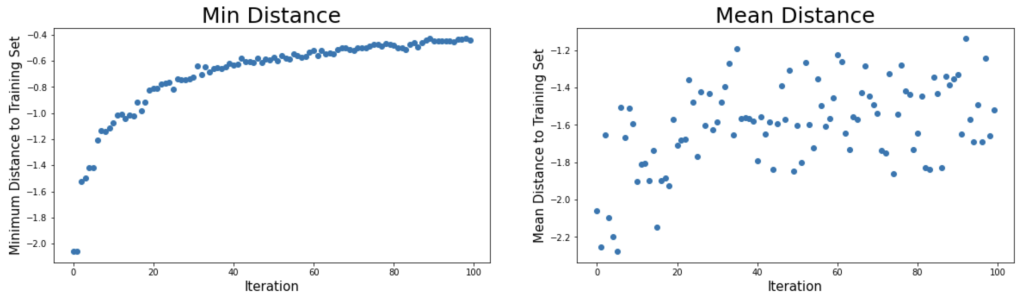


Figure 6.4 Comparing minimum Euclidean distance against mean Euclidean distance as a diversity metric. Here minimum distance is used to select the next point in the set and both metrics of those points are displayed. We can see that there is little correlation between the two metrics indicating they provide different information. The R^2 between these two metrics on these points is just 0.37. The Spearman Rho, rank-correlation, is also low at 0.44.

To further compare the minimum and mean distance metrics, the analysis was flipped, such that mean distance was used to select new points and both metrics were recorded on the selected points. These results are shown in Figure 6.5. Here it becomes obvious that mean distance is not a good metric since the minimum distance metric indicates that we are repeatedly selecting points already in the set. This is shown by the consistent minimum distance value of 0 after around 10 iterations. This result led to mean distance being thrown out as a potential choice of metric.

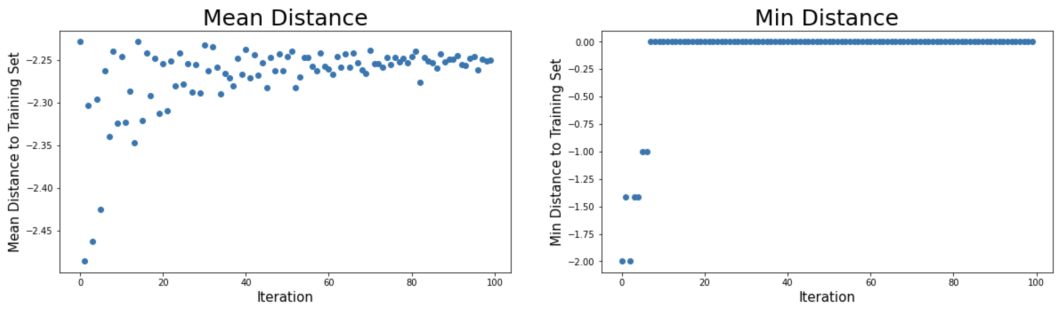


Figure 6.5 Comparing mean Euclidean distance against minimum Euclidean distance as a diversity metric. Here mean distance is used to select the next point in the set and both metrics of those points are displayed. We can see that when mean distance is used to select new points, we get many points with a minimum distance of 0. This indicates that we are very frequently reselecting points already in the set. This shows that minimum distance is a better metric than mean distance.

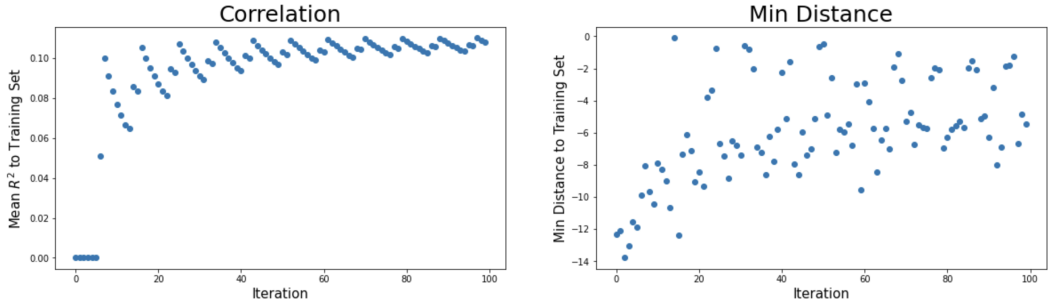


Figure 6.6 Comparing mean point correlation against minimum Euclidean distance as a diversity metric. Here minimizing mean correlation is used to select the next point in the set and both metrics of those points are displayed. We can see that there appears to be a weak positive correlation between the two, indicating that they provide some of the same information but are not the same, so may have different advantages. It is also promising that the minimum distance shows that we are not reselecting points already in the training set. Comparing the metrics for these points we get an R^2 of 0.35 and a Spearman Rho of 0.33.

Minimum distance and correlation were also compared to determine if they provide unique measures of diversity. The results are shown in Figure 6.6. For this analysis, lack of correlation to the training set was used to select new points and both metrics were recorded. This analysis was slightly different than the previous ones since for this problem the points were embedded in a 10-dimensional space instead of just 3. The results show that the two metrics do provide unique information since an R^2 value of 0.35 and a Spearman Rho value of 0.33 were recorded, which are both low. Since these metrics were determined to provide unique information without any clear flaws both were included to be explored, with the one limitation that correlation as a diversity metric could not be used on problems of less than 3 dimensions.

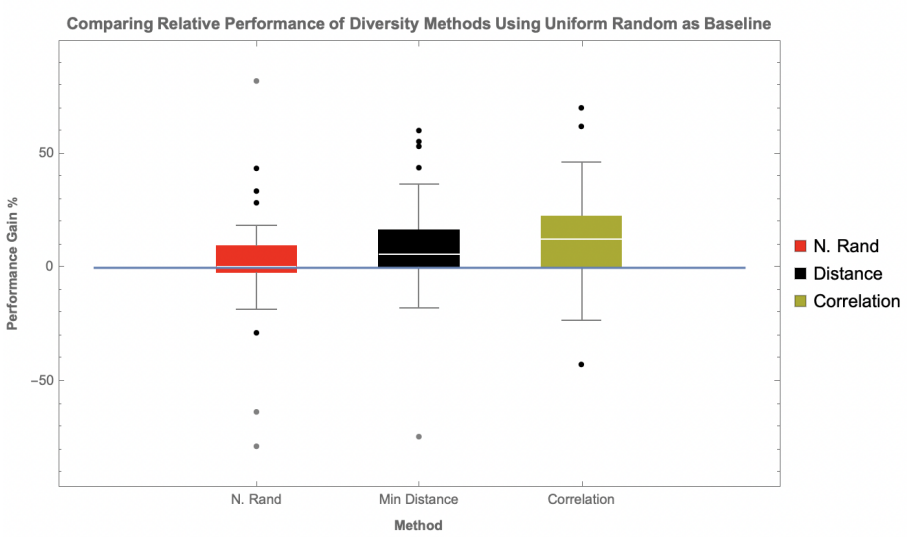


Figure 6.7 Comparing Relative Performance of Diversity Methods Using Uniform Random Selection as Baseline. Shown here are the performance differences of both the AL diversity methods compared to uniform random selection as the baseline (blue line) and normally distributed random selection (red distribution). We see that using minimum distance (green distribution) performs consistently better than the baseline and correlation (blue distribution) works best as a diversity metric. The drawback with using correlation as the diversity metric though is that it requires problems with more than two dimensions, so the problems with two dimensions are ignored when using correlation. The distributions represent the median performances of 100 independent runs across all test problems.

The results of comparing the different data diversity-based active learning methods are summarized in Figures 6.7 and 6.8 and the full results are shown in Table C.2 in Appendix C. Figure 6.7 uses uniform random sampling as the baseline for comparison, shown as the blue line. We again include normally distributed random sampling for comparison as the red distribution. We can see that both diversity metrics have better performance than uniform random sampling, on average requiring fewer training points to find a solution. We also see that correlation as a diversity metric performs best, often requiring the least number of training data points to find a solution. Correlation does have the disadvantage, though, of not working on the problems with just two dimensions. Those two problems are not represented in the correlation bar in the chart since they are not applicable.

Figure 6.8 shows the number of cases where each method either outperformed or underperformed when compared to uniform random sampling. We see again that correlation has the best

performance. This indicates that not only does correlation lead to requiring fewer training points on average, but also indicates that it most consistently requires fewer points. We see that distance as a metric requires fewer points than uniform and random sampling, but is not as consistent as correlation.

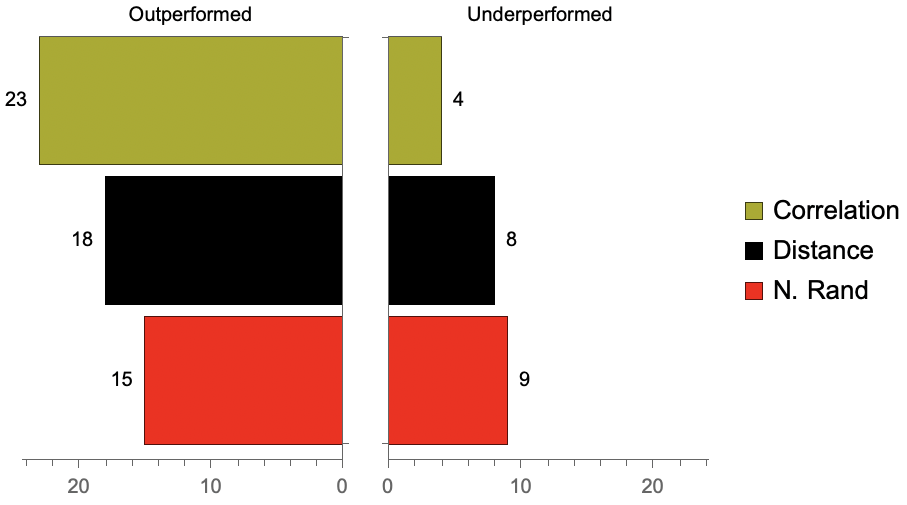


Figure 6.8 Comparing Performance of Diversity Methods Against Uniform Random Selection. Each method is compared to uniform random sampling and the number of times that the method outperforms and underperforms is reported. The number of times each method outperforms is shown on the left and the number of times each method underperforms is shown on the right. Outperforms means that a method used fewer points than uniform random sampling. Underperforms means it required more points. Ties are not counted but can be easily determined by taking the difference of 35 and the two values reported. The results show that correlation performed best, underperforming the fewest times and outperforming the most.

6.4.3 Comparing Diversity, Uncertainty, and Pareto Optimization of Both

Next, we explore how the performance compares when using uncertainty and diversity together to see if there are benefits to considering both for selecting training data with AL compared to just uncertainty or diversity alone. For this comparison, we selected one diversity metric and one uncertainty metric. For the uncertainty metric, we chose differential entropy since it was shown to be the best performing metric in Figure 6.2. For the diversity metric, we chose minimum distance. Although it didn't perform best, it is most versatile since it isn't restricted to problems with more than 2 dimensions. For the combination method, we used a Pareto optimization to find the points with the best trade-off of both the uncertainty and diversity metrics from 10,000

randomly generated points each iteration. From the Pareto front of points that are non-dominated in those two objectives, we ordered them based on their uncertainty score and selected the median point. Note that sorting based on uncertainty is just the reverse order of a sort by diversity, so which objective you choose to sort by shouldn't have a significant impact. The only impact would be on cases where an even number of points are on the front so the point you select isn't the true median but rather one of the points near the median. When this occurs, we round down to select the median point, which would give a slight bias toward uncertainty. By selecting the median point we are attempting to choose a point that has a relatively good balance between the two objectives.

The results of this comparison are shown in Figures 6.9 and 6.10 with the results from each problem shown in Appendix C in Table C.3. Again in Figure 6.9, we use uniform random sampling as the baseline (blue line) and include normally distributed random sampling for comparison. The results show that all three methods work better than the baseline and normally distributed random sampling. Using the uncertainty metric, differential entropy, works slightly better than using the distance metric, minimum distance. We also see that there is a benefit to combining both metrics using the Pareto optimization since we see an improvement in the upper quartile of performance. It is also interesting to note, as can be seen in Figure 6.10, that the diversity metric alone performed worse than uniform random sampling in 8 of the 35 cases, whereas the uncertainty approach and the Pareto approach only performed worse in 4 of the cases, demonstrating that the uncertainty and Pareto approaches offer more consistent improvements. This indicates that it is important to consider the current models to help guide the AL process. This makes sense since the goal is to select training points that will best inform the current model population, using only diversity doesn't consider the current state of models, so it is less likely that the training points selected will most inform those models. Statistical significance tests were also performed and the number of cases determined to be statistically significant are shown in the darker regions in the figure. The Mann-Whitney test was used to test for significance and a threshold of 0.05 was used. The Pareto approach was found to be statistically significant in 18 of the 20 cases where the Pareto approach outperformed.

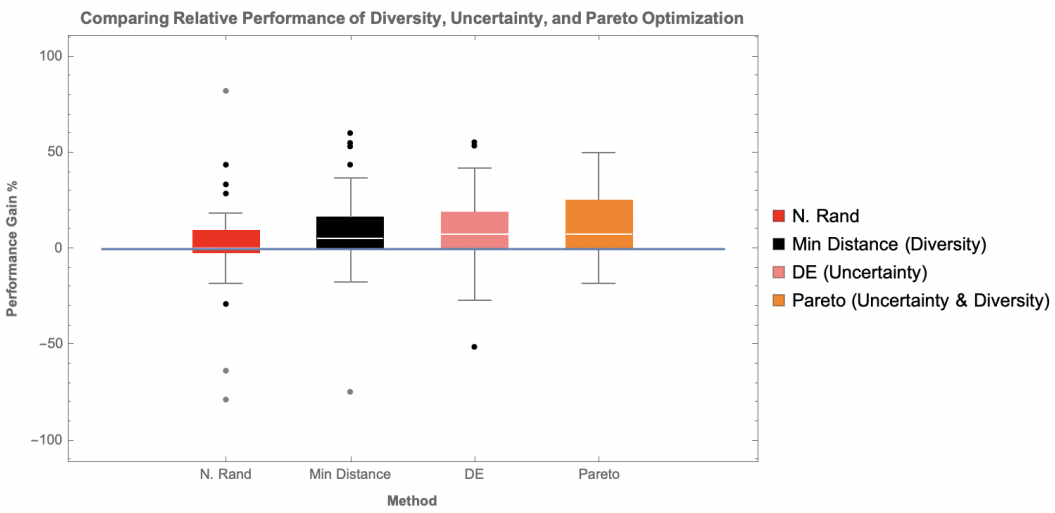


Figure 6.9 Comparing Relative Performance of Diversity, Uncertainty, and Pareto Optimization Using Uniform Random Selection as Baseline. Shown here are the performance differences of AL diversity, uncertainty, and Pareto methods compared to uniform random selection as the baseline (blue line) and normally distributed random selection (red distribution). We see that using the diversity metric, minimum distance (green distribution), performs consistently better than the baseline and the uncertainty metric, DE (blue distribution), performs a bit better than the diversity method. When using a Pareto optimization of both diversity and uncertainty we get even better performance. The distributions represent the median performances of 100 independent runs across all test problems. For completeness, there is a single point around -150 for the Pareto approach.

Looking at the results, there are two instances where the Pareto approach performed considerably worse than the uncertainty and diversity approaches. Those are equations 9 and 71. Table C.3 in Appendix C shows that the combined method performs worse than focusing alone on either diversity or uncertainty for those two problems. This is likely a result of equations 9 and 71 being higher dimensional problems with 6 and 5 dimensions, respectively, so the 10,000 randomly generated points don't sufficiently fill the search space to find points with high values for both uncertainty and diversity.

Equation 71 was further explored to see if sampling additional points improved the performance when using the combined diversity uncertainty approach and to verify that sparse sampling was at least part of the issue as suspected. Equation 71 was retested using 100,000 randomly sampled points to search for the best trade-off between diversity and uncertainty. When using 100,000 points the median number of points required to solve the problem decreased to 42 points from 50.5, confirming that better sampling of the space improves the performance in this higher dimensional

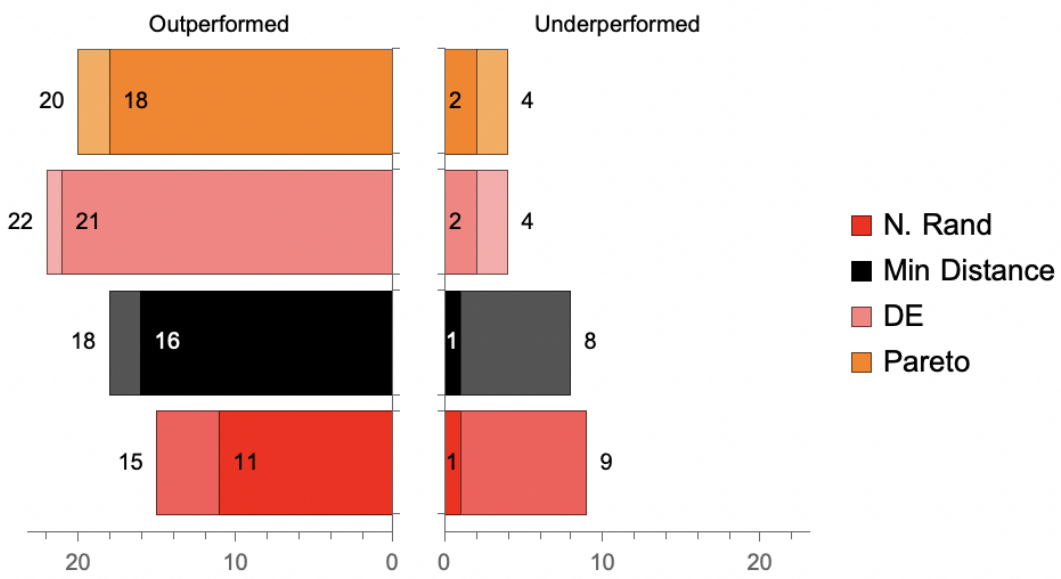


Figure 6.10 Comparing Performance of Diversity, Uncertainty, and Pareto Optimization Against Uniform Random Selection. Each method is compared to uniform random sampling and the number of times that the method outperforms and underperforms is reported. The number of cases where the differences are statistically significant is shown in the darker regions. The number of times each method outperforms is shown on the left and the number of times each method underperforms is shown on the right. Outperforms means that a method used fewer points than uniform random sampling. Underperforms means it required more points. Ties are not counted but can be easily determined by taking the difference of 35 and the two values reported. The results show that DE, the uncertainty method works best. The Pareto approach ties for the least number of underperforming cases, matching DE, and outperforms between DE and Min. Distance. Statistical significance was determined using the threshold of 0.05 with the Mann-Whitney test.

problem. The median performance of 42 points is still worse than either of the uncertainty or diversity approaches, so more points could be used, but increasing the number of points beyond 100,000 begins to make that search rather expensive. Rather than randomly sampling the points then selecting the Pareto front from those points, an alternative optimization method, such as NSGA II [Deb et al., 2002], could be used in future studies which might be cheaper and likely more effective.

6.4.4 Additional Benchmark Problems

To further test the Pareto AL approach, we selected two problems from a more recent benchmark set, SRBench [La Cava et al., 2021]. One that is on the easier side for StackGP and one that is a bit more challenging. The easier problem selected was the van der Pol oscillator problem, referred

to as "strogatz_vdp1" in SRBench. The equation for the van der Pol oscillator problem that we are trying to rediscover is $x' = 10 * (y - (1)/(3) * (x^3 - x))$. The more challenging problem was the bar magnet problem, referred to as "strogatz_barmag1" in SRBench and the equation for the bar magnet problem that we are trying to rediscover is $x' = 0.5 * sin(x - y) - sin(x)$. As with the previous problems, we performed each experiment 100 times and computed the median number of points to find the solution. The results of those experiments are shown in Table 6.2. We can see that the Pareto approach performs significantly better than randomly sampling from a normal distribution and performs about 27.8% better than randomly sampling from a uniform distribution on the bar magnet problem. The performance gains over the normal and uniform distributed samplings are statistically significant considering a threshold of 0.05 using the Mann-Whitney test. We computed a p-value of $3.490*10^{-11}$ when comparing to the normal distribution and $6.481*10^{-6}$ when comparing to the uniform sampling. We also see better performance on the van der Pol oscillator, although since it was an easy problem there isn't as much opportunity for improvement, so we only see a reduction of a few points. The performance gains over the normal and uniform distributions are again statistically significant with a p-value of $2.51*10^{-7}$ when compared with the results from using normally distributed sampling and a p-value of $4.008*10^{-13}$ when compared with the results from using uniform random sampling.

SRBench Problem	N. Ran Data Pts.	U. Ran Data Pts.	Pareto AL Data Pts.
Bar Magnet #1	51	18	13
Van der Pol Osc. #1	10	9	7

Table 6.2 Shown are the median numbers of points needed to solve each equation. A total of 100 independent trials were performed for each equation. We compare the active learning method that uses both diversity and uncertainty and compare the performance against random sampling on two problems from the SRBench.

6.5 Conclusion

Both uncertainty and diversity metrics for active learning were explored to see how each metric impacts the success of active learning in genetic programming. As well, a Pareto approach was

defined that allows both diversity and uncertainty to be considered for active learning. Of the uncertainty approaches, it was observed that differential entropy performed best. It was also observed that relative uncertainty functions did not perform well. When using differential entropy it was found that performance could be boosted by using differential evolution as the optimizer over Scipy Optimize's minimize function. This indicates that the search space is not convex and requires a good optimizer to find solutions with high uncertainty.

When comparing the data diversity methods, it was found that correlation performed better than minimum Euclidean distance. Although correlation worked better, it does not work on cases with 2 dimensions or less. Thus, minimum Euclidean distance was selected for the Pareto approach. Future implementations may default to using minimum Euclidean distance for all cases with 1 or 2 dimensions and using correlation for higher dimensional problems. Mean distance was considered, but determined to be uninformative due to its frequency of identifying repeat points.

When comparing the Pareto approach which used both differential entropy and minimum Euclidean distance to differential entropy, minimum Euclidean distance, uniform random selection, and normally distributed random selection, it was found that differential entropy worked best, with the Pareto approach performing between differential entropy and minimum Euclidean distance. Looking at individual problems, there were a few cases where the Pareto approach actually worked better than both differential entropy and minimum Euclidean distance on their own, indicating potential benefits of combining the two approaches. For the cases where the Pareto approach did not work as well, it was identified that the multi-objective optimization strategy may have been at fault since it relies on randomly generating N points and selecting the median value in the Pareto front. Better methods such as NSGA-II could be explored in future studies to see if improved optimization methods lead to better active learning performance.

Overall, it was found that active learning can be efficiently utilized with genetic programming to reduce training data requirements. In practice, this would be useful to apply in scenarios where collecting data or labelling data is expensive, and model training is relatively cheap. In these scenarios, active learning could be used to guide data collection and labelling so that good models

can be arrived at using as few data points as possible. This application has the potential to accelerate data-driven research since it could lead to finding solutions with fewer resources in less time.

6.6 Acknowledgments

Computer support by MSU's iCER high-performance computing center is gratefully acknowledged.

6.7 Code Availability

The code for StackGP with active learning can be found here: https://github.com/hoolagans/StackGP

Part III

Classification

Classification is another common application of machine learning, where rather than finding a mathematical function to fit data, the goal is to find a model that fits data with several discrete values (classes) as labels. In general, for classification tasks the input data can be a mix of categorical and numeric data, but in this work, we focus on input data that is purely numeric and is extracted from images. Two classification methods are explored, decision tree GP (DT-GP) and SEE-Segment. Both approaches are used for image segmentation tasks, where the classification is binary such that each pixel is either labelled as foreground or background. Foreground is the part of the images that is interesting and background is what we want to remove or ignore. DT-GP is then tested further on the task of labelling cells in an image as either co-transfected or not. Co-transfected means that a cell is expressing two different proteins of interest that were inserted by a researcher.

This section focuses on how active learning can be applied to population-based ML systems on classification tasks. We first describe both DT-GP and SEE-Segment in detail and explore how active learning was applied with both systems to improve model development. We then apply active learning with both systems and discuss the results.

CHAPTER 7

DT-GP

Decision tree genetic programming (DT-GP) is a system I developed that can evolve decision trees to solve binary classification tasks. In this work, the trees operate on numeric values and vectors to produce binary true and false classifications. This allows the system to work for image segmentation, where the goal is to take the numeric values in each pixel and classify them as foreground or background, as well as allowing the system to take in numeric vectors, representing whole or partial images and assign them to a class. The implementation of this system is described in detail in this chapter and it is applied to image analysis problems with active learning in Chapter 9.

7.1 Model Form

Image segmentation and object classification were performed using an implementation of decision tree GP (DT-GP), where decision trees are evolved to consider which pixels are foreground or background for segmentation and to identify which class an object belongs to for object classification tasks. The decision trees can utilize 3 types of operators: boolean operators, (in)equality operators, and numerical operators. Boolean operators can take in boolean values and return boolean values. Inequality and equality operators take in numeric values and return boolean values. Numeric operators can take in numeric vectors and return numeric scalars.

The boolean operators available are And, Or, Not, $Nand$, Nor, and Xor. The inequality operators available are >=, >, <=, <, ==, and ! =. The numeric operators are $average$, $median$, max, min, $difference$, $range$, $standardDeviation$, $getRed$, $getGreen$, and $getBlue$. The last three operators listed simply grab the red, green, or blue values from pixels. The vector operations operate on a specific color stream when given an image and they operate on a pixel vector when operating on individual pixels. The models can also contain randomly generated constants as floating point values from 0 to 1 since the RGB pixel values are stored as values from 0 to 1.

To make the use of these operators meaningful, a hierarchical structure is enforced in the trees such that the boolean operators can only operate on inequality operators, and inequality operators

can only operate on numeric operators and constants. Numeric operators are then restricted to the lowest levels of the decision trees where they can operate on numeric data. Trees can only be initialized in this form and then this form is enforced throughout evolution by restricting how mutation and crossover can be performed.

An example of a tree generated to segment the KOMATSUNA plant data Uchiyama et al. is shown in Figure 7.1. Note that although this tree happens to be balanced, it is not enforced, so trees generated by DT-GP can potentially be heavily skewed.

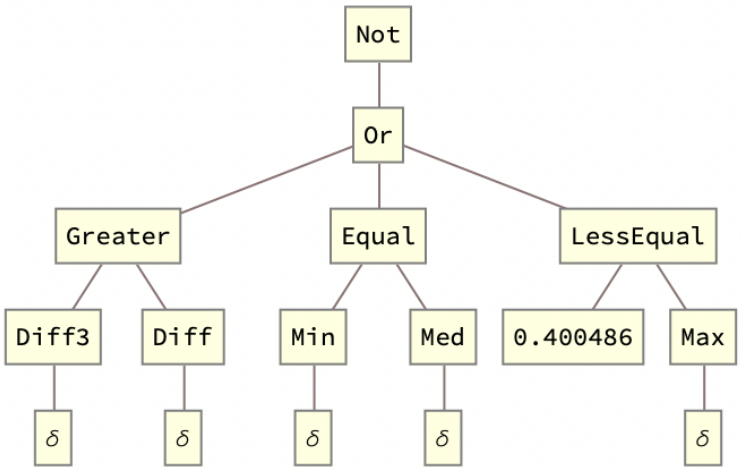


Figure 7.1 Shown here is an example tree generated to segment the KOMATSUNA data set. Here delta is a placeholder for the data. For this tree, we can see that the top two layers are boolean operators, below that we have one layer of inequality operators, and the bottom layers consist of numeric operators or numeric values.

7.2 Genetic Operators

Subtree crossover, subtree mutation, and point mutation were used as the variation operators. Point mutation is simple, just replace an operator or value with an operator or value of the same type. For example, if a boolean operator is selected for point mutation, then only a boolean operator can be chosen to replace it. A visual representation of a point mutation is shown in Figure 7.2. Subtree mutation is performed by selecting a position in a tree and producing a new randomly generated subtree to replace that position, ensuring that the new tree is valid considering the operators above where the subtree is being added. An example of subtree mutation is shown in Figure 7.3. For

subtree crossover, to ensure models created are meaningful, an operator of the same type is chosen from each parent tree and the subtrees from those points down are swapped to produce two new child models.

Mutation and crossover are exclusive, meaning a child model in one generation can be produced by either mutation or crossover, but not both. The mutation and crossover rates are shown in Table 7.1 along with the rest of the parameters used to run DT-GP.

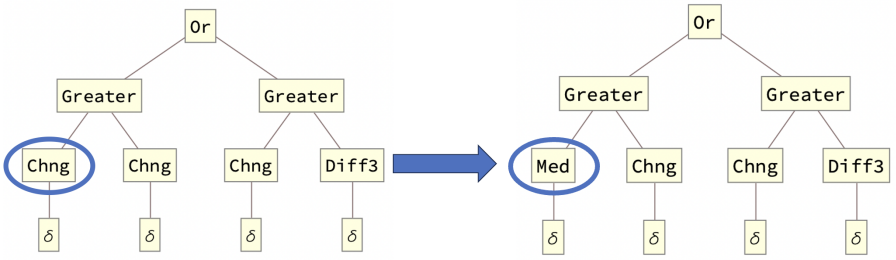


Figure 7.2 Shown here is an example tree and the result of applying a point mutation. The resulting point mutation replaced the "Chng" operator with the "Med" operator.

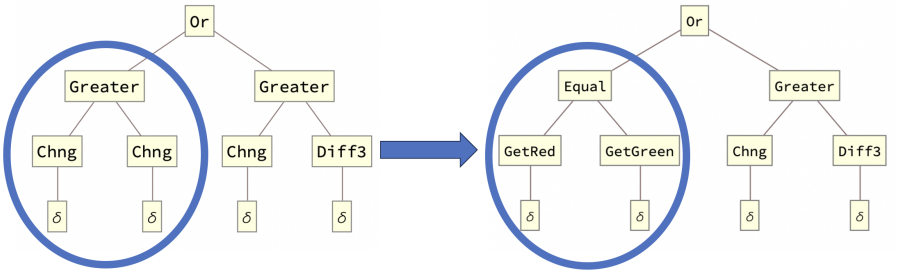


Figure 7.3 Shown here is an example tree and the result of applying a subtree mutation. The resulting subtree mutation replaced the left side of the tree with a new subtree.

7.3 Selection/Fitness

For classification tasks, we utilize a simple fitness function that is just the percent of correctly classified inputs divided by the total number of inputs, with the one trick being that we do not care which label the model assigns to each class. For example, if the true class labels are A and B, but a model assigns the labels B and A, instead of a 0% fitness we get a 100% fitness. This is achieved by defining the fitness function as $Max(c/t, 1 - c/t)$, where c is the number of correctly classified cases and t is the number of total cases. Since we are operating on binary classification

Parameter	Setting
Mutation Rate	30
Crossover Rate	40
Spawn Rate	10
Elitism Rate	20
Crossover Method	Subtree
Mutation Method	Point & Subtree
Tournament Size	5
Population Size	30
Generations	100
Max Tree Depth	6

Table 7.1 DT-GP & Active learning Parameter Settings.

problems, this fitness function leads to a guaranteed minimum fitness of 50% on the training set and also shrinks the search space, since each model is considered in two states, the original state and the state of the model if the *Not* operator were applied to it simultaneously. Post evolution, we can then assign the correct model state to each model in the population. This is analogous to the alignment step done at the end of evolution when using StackGP with the correlation fitness function for regression tasks.

For selection, DT-GP utilizes standard tournament selection, where each tournament considers 5 randomly selected models and returns the single model with the best fitness. All models selected for mutation or crossover are selected via tournament selection.

Table 7.1 shows the parameters used to run DT-GP with AL-GP.

7.4 Active Learning Implementation for DT-GP

Active learning was used with DT-GP to select training images iteratively to maximally inform model populations. Uncertainty was quantified by measuring disagreement between the models in an ensemble. To do this, each model in the ensemble was used to generate a predicted segmentation pattern on each potential image to be added to the training set. For each image, every pair-wise pixel difference was computed for each model in the ensemble's predicted segmentation pattern. This difference can be summarized as the total number of pixels where the two models disagree on its classification. The pair-wise differences of all models in the ensemble are then averaged. The

image that returns the largest uncertainty value is then selected and added to the training set. This approach is described in detail in Algorithm 7.1 Evolution then resumes using the now expanded training set.

The model ensemble is selected in a way that attempts to capture the diversity of the population while also containing primarily high-quality individuals. This is done by selecting the top 10 models from the population that have unique fitness values on the training set.

procedure UNCERTAINTY(HoF,$Image$)
    $Uncertainties \leftarrow []$ ▷ Initialize uncertainties list
    $Pairs \leftarrow GeneratePairs(ensemble)$ ▷ Generates all pairs of models in HoF
    for i 1 to len(Pairs) **do**
        $Prediction1 \leftarrow EvaluateModel(Pairs[i][0], Image)$ ▷ Generate prediction of first model in pair
        $Prediction2 \leftarrow EvaluateModel(Pairs[i][1], Image)$ ▷ Generate prediction of second model in pair
        $UncertaintyVal \leftarrow Total(Difference(Prediction1, Prediction2))$ ▷ Compute Pixel/Object Differences and Total
        $Uncertainties \leftarrow Uncertainties.append(UncertaintyVal)$ ▷ Append to list
    end for
    $MeanUncertainty \leftarrow Mean(Uncertainties)$ ▷ Compute mean of uncertainties
    Return $MeanUncertainty$
end procedure

Algorithm 7.1 Uncertainty Computation DT-GP.

The application of AL-GP with DT-GP to image analysis tasks is explored in Chapter 9.

7.5 Code Availability

The code for DT-GP can be found here: https://github.com/hoolagans/DTGP/tree/main

CHAPTER 8

SEE-SEGMENT

SEE-Segment is a tool developed by the SEE-Insight group that uses a genetic algorithm to search a space of computer vision algorithms and parameters associated with those algorithms. The goal is to find a model that properly segments an image or set of images. The intended use of this system is to be part of a researcher's image segmentation workflow whereas the user is manually segmenting images, the tool will search for segmentation algorithms that will learn to replicate the manual segmentation patterns and hopefully automate the segmentation of images after being trained on a few. Active learning could benefit this workflow by suggesting which images should be manually labelled, such that maximal information is gained from each image which could lead to fewer images being labelled until the task can be automated.

8.1 SEE-Segment Implementation

SEE-Segment was used to evolve a population of image segmenters. Each individual model consists of a segmentation strategy and 8 parameters. The parameters and choice of strategy are optimized through generations. While each individual contains 8 parameters not all parameters are used by all strategies. Mutation and crossover are used as the variation operators. Tournament selection is utilized for selection. All models selected from tournament selection then have a 90% chance of crossover and then from those offspring, there is a 90% chance of a mutation. Elitism is also utilized by guaranteeing the Hall of Fame models (top 10 genotypically unique models) are preserved across generations. The parameters used to run SEE-Segment are shown in Table 9.1. Spawn rate refers to the number of new models introduced in a generation. The number spawned each generation is the number required to bring the population size back up to 100 after including the hall of fame models and the unique individuals produced from mutation and crossover. Tournament size, population size, and generations are the only parameters modified from the default settings. Those settings were chosen since they allowed for sufficient model development while keeping computation time reasonable in each iteration of learning.

This instance of SEE-Segment is modified from the original SEE-Segment since I introduced

Parameter	Setting
Mutation Rate	90
Crossover Rate	90
Spawn Rate	100-HoF-UniqueMods
Hall of Fame Size	10
Tournament Size	4
Population Size	100
Generations per AL Iteration	100

Table 8.1 SEE-Segment Parameter Settings.

tournament selection as the method for selecting which models will undergo mutation and crossover. I also introduced a function that limits the number of duplicates that can exist in any population. I set the limit to 1 so that no duplicate genotypes can exist in the population. Previous to these edits, I frequently encountered populations that would converge to a single suboptimal model very quickly, since only the Hall of Fame models were selected for reproduction which led to populations quickly losing all diversity. These updates were recently adopted by the main branch of SEE-Segment.

8.2 AL Implementation for SEE-Segment

Two active learning methods were implemented that vary in how they select the ensemble. The first method simply uses SEE-Segment's Hall of Fame, which consists of the top 10 genotypically unique models. The second method goes further to ensure models are unique by selecting the top 10 phenotypically unique models. These ensemble selection methods differ from the original AL-GP approach since here we start with only a single image. The original approach relies on generating diverse data clusters and selecting models that best fit each data cluster. If that approach was utilized here, we would initially get an ensemble of one model, which would lack any sort of uncertainty metric.

Using the selected ensemble we compute the average pairwise disagreement of all the models in the ensemble on each potential image. The image with the maximum average pairwise disagreement is selected for labelling and added to the training set. In the event of a tie, the first image found with the max value is selected. An overview of the active learning algorithm is shown in Algorithm 8.1 and the method for computing uncertainty is shown in Algorithm 8.2. The pairwise uncertainty

on an image between two models is computed by utilizing SEE-Segment's fitness function where instead of supplying the fitness function with a model's predicted segmentation mask and a true mask, the predicted masks from both models are supplied. This leads to a reasonable measure of how different two model's predicted masks are.

The type of active learning implemented to work with SEE-Segment would be classified as pool-based active learning since it works by scanning over all un-labelled images in a data set and selecting a single image to label and add to the training set each iteration. To select an un-labelled image that will be informative to the current model population we select an image that generates significant disagreement amongst the top-performing models in the population. The top performing models are then stored in the evolver's Hall of Fame which consists of the top 10 models discovered so far during evolution. Each model in the Hall of Fame is guaranteed to have a unique genotype, but this does not guarantee that the phenotype will be unique. Using the Hall of Fame models we compute the average pairwise disagreement of all the models on each potential image. The image with the maximum average pairwise disagreement is selected for labelling and added to the training set. In the event of a tie, the first image found with the max value is selected. An overview of the active learning algorithm is shown in Algorithm 8.1 and the method for computing uncertainty is shown in Algorithm 8.2. The pairwise uncertainty on an image between two models is computed by utilizing SEE-Segment's fitness function where instead of supplying the fitness function with a model's predicted segmentation mask and a true mask, the predicted masks from both models are supplied. This leads to a reasonable measure of how different two model's predicted masks are.

```
1: TrainingData ← RandomImage                                  ▷ Select 1 Random Image
2: Evolver ← GeneticSearch.Evolver(TrainingData)               ▷ Initialize evolver
3: evolver.run()                                               ▷ Evolve models with initial data
4: while i ≤ iterations do                                     ▷ While max iterations not reached
5:     HoF ← Evolver.hof                                       ▷ Extract hall of fame
6:     SelectedImage ← MaximizeUncertainty(HoF, Data)          ▷ Find image that maximizes uncertainty
7:     TrainingData ← Append(TrainingData, SelectedImage)      ▷ Add new image to training data
8:     Evolver.UpdateData(TrainingData)                        ▷ Update training data in evolver
9:     evolver.run()                                           ▷ Evolve models with new data
10: end while
```

Algorithm 8.1 Active Learning Process for SEE-Segment.

```
1: procedure UNCERTAINTY(HoF,Image)
2:     Uncertainties ← []                                  ▷ Initialize uncertainties list
3:     Pairs ← GeneratePairs(HoF)                          ▷ Generates all pairs of models in HoF
4:     for i 1 to len(Pairs) do
5:         Prediction1 ← EvaluateModel(Pairs[i][0], Image)     ▷ Generate prediction of first model in pair
6:         Prediction2 ← EvaluateModel(Pairs[i][1], Image)     ▷ Generate prediction of second model in pair
7:         UncertaintyVal ← FitnessFunction(Prediction1, Prediction2)   ▷ Compute uncertainty
8:         Uncertainties ← Uncertainties.append(UncertaintyVal)         ▷ Append to list
9:     end for
10:    MeanUncertainty ← mean(Uncertainties)               ▷ Compute mean of uncertainties
11:    Return MeanUncertainty
12: end procedure
```

Algorithm 8.2 Uncertainty Computation SEE-Segment.

The application of AL-GP with SEE-Segment to image segmentation tasks is explored in Chapter 9.

8.3 Code Availability

The code for SEE-Segment can be found here: https://github.com/see-insight/see-segment

CHAPTER 9

AL-GP IN CLASSIFICATION TASKS

Here we explore the efficacy of active learning in genetic programming (AL-GP) for image processing tasks using two new population-based machine learning systems, decision tree genetic programming and SEE-Segment as described in Chapters 7 and 8. We explore how active learning can be used to improve the rate and consistency of finding good models using few data points. The importance of diversity in ensembles for AL-GP is also explored by varying the definition for diversity when performing active learning with SEE-Segment. Finally, we demonstrate how AL-GP was deployed in a research setting to help automate and accelerate progress by guiding labelling of training samples (human cells) to inform the development of classification models which were then used to automatically classify cells in video frames.

9.1 Introduction

Image analysis is the process of extracting useful information from image data. This extracted information can then be used to study systems captured in the image data. Image analysis is broadly applied from medical imaging to computer vision [Shen et al., 2017] [Wäldchen and Mäder, 2018]. In medical imaging, the image data will often come from magnetic resonance imaging (MRI), positron emission tomography (PET), computerized tomography (CT), x-rays, etc [Shen et al., 2017]. For example, in [Suk and Shen, 2015] the authors are able to use MRI and PET image data with deep learning methods to improve the success rate of identifying Alzheimer's disease. Image analysis involves extracting useful information from image data, which is generally rich with information, but can also contain significant noise. Segmentation is a specific step in image analysis where the features of interest are isolated and background information (noise) is removed.

As was introduced in Chapter 2, active learning is the field of machine learning focused on selecting training data that will maximally inform model development and uncertainty sampling is a type of active learning where model uncertainty is used to guide data selection.

In Chapter 2, I introduced the three main classes of active learning: pool-based, stream-based, and membership query synthesis. In this chapter, I use pool-based methods of active learning. As

a reminder, pool-based methods work when you have an existing set of unlabelled data and the task of active learning is to search over that pool of data to select the specific samples that are predicted to be most informative. This is implemented in this chapter by searching over the pool of samples and returning one most informative sample in each round of active learning.

Uncertainty-based active learning is used in this chapter, and while genetic programming models individually lack statistical properties to compute uncertainty, the model populations present in GP can be utilized to quantify uncertainty across the diverse models within a population. We have previously applied active learning to genetic programming in symbolic regression tasks using active learning in genetic programming (AL-GP) with a stack-based genetic programming system (StackGP) [Haut et al., 2022] and discussed that work in Chapter 6.

This general strategy of utilizing the model populations in symbolic regression GP tasks to select training data to maximally inform evolution seemed generalizable to GP and evolutionary computation, so in this work, we demonstrate how AL-GP can be implemented and applied to other evolutionary computation methods and various image analysis problems to accelerate the development of segmentation and object classification models. These models can then be used to aid research in various fields reliant on image data.

The SEE-Insight project is an open-source framework to accelerate the biggest bottleneck in Scientific Image Understanding, which is manual image annotation. SEE-Segment is the first tool developed for SEE-Insight and consists of an evolutionary machine learning approach that utilizes a genetic algorithm to select a computer vision algorithm and optimize parameters for an image annotation task (image segmentation). SEE-Segment will work with a Graphical User Interface (GUI) allowing researchers to upload their image datasets and then incrementally annotate their images. The image annotations are used to test scientific hypotheses or as a first step to feeding into a data-driven model such as a neural network. Because annotating images can be slow and tedious, while researchers are interfacing with the GUI the SEE-Segment system is simultaneously searching this grammar (aka "algorithm space") to find automated methods that can reproduce their manual workflows. This search is happening "in the background" on large-scale systems. Given the

complexity and size of the search space, there is no guarantee that it will converge to a reasonable solution. However, if a good algorithm is found then suggestions are passed to the researcher to help speed up their annotation process. In the best case, a fully automated algorithm is identified that can reproduce their manual annotation. In the worst case, SEE-Segment will not take any longer than manually annotating images without the discovery tools.

This application is an instance of a Combined Algorithm Search and Hyperparameter (CASH) problem and uses a genetic algorithm to search for an algorithm (and hyperparameters). Although the space is nondifferentiable and highly heterogeneous, preliminary results are promising. By using a well-defined image grammar and genetic algorithms as the core search tool, results of the machine learning are highly human interpretable. This allows the system to "generate code" that can be used for teaching as well as copy-and-pasted to a researcher's own program.

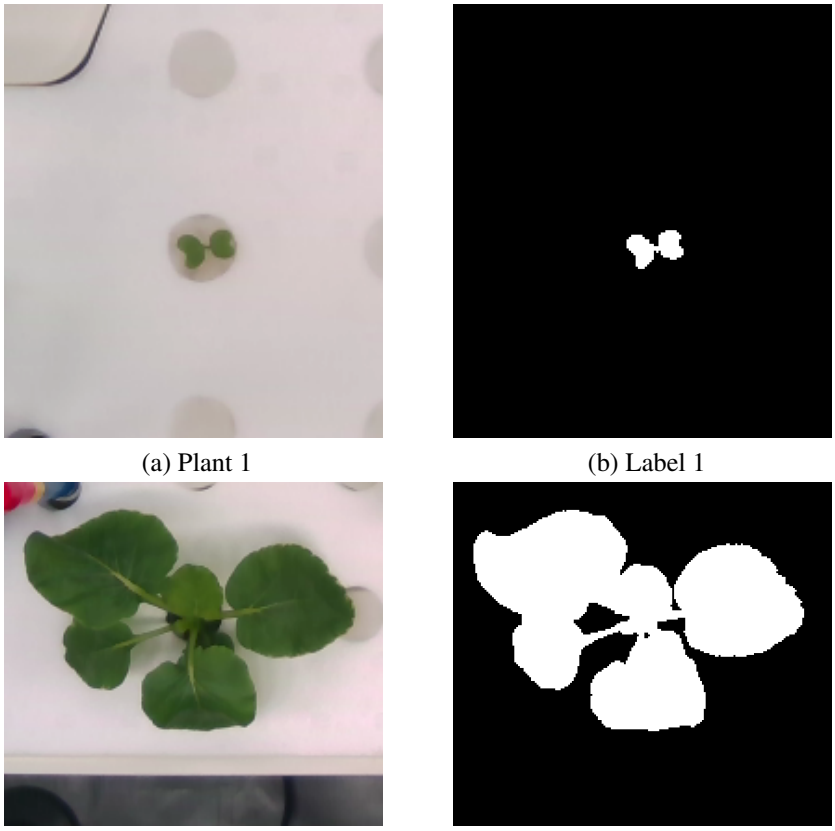
Decision tree GP (DT-GP) is a GP system that evolves decision trees and was developed as part of this work specifically to solve the problem of cell classification but is generalizable to classification tasks.

In this work, we explore the efficacy of AL-GP in two different population-based ML systems and then demonstrate how AL-GP can be applied in a research setting to accelerate progress in scientific studies.

9.2 Data Sets

9.2.1 KOMATSUNA

To benchmark the active learning methods in both systems, the KOMATSUNA [Uchiyama et al.] data set was used since it is a fairly simple segmentation problem and has ground-truth labels available. The KOMATSUNA data set contains 300 images of plants where the provided labels are the segmentation patterns that identify the plant from background data. The KOMATSUNA data set is ordered and tracks plants over time as they grow. Each set of 5 consecutive images is taken within the same day so the images are substantially similar within those sets. Example images from the KOMATSUNA data set are shown in Figure 9.1 to demonstrate how the sizes of plants vary in the set and also how the camera angle and location can vary.



(a) Plant 1 (b) Label 1

(c) Plant 2 (d) Label 2

Figure 9.1 Example image (a) from KOMATSUNA dataset with its corresponding label (b). The label is the true segmentation mask. A second example image (c) and its label (d) are shown to demonstrate the diversity of images in the dataset.

9.2.2 Sky Segmentation

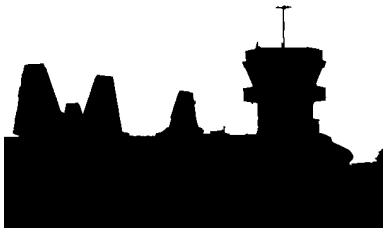
The sky data set [Alexandre and Miranda] contains 60 images of the sky where the labels provide the segmentation patterns that identify the sky from other objects. Of those 60 images, 4 had to be removed since the supplied segmentation patterns are not correct since they all segment out the entire image. Those are images '0001' to '0004' in the set.

The sky data set contains a selection of images where the feature of interest is the sky and other objects obstructing the view of the sky are to be segmented out. Every image contains a plane either in the sky or near the ground, in which case the ground also needs to be segmented out. Examples from the data set are shown in 9.2 to demonstrate an example of two images and their true segmentation patterns.

(a) Sky 1

(b) Label 1

(c) Sky 2



(d) Label 2

Figure 9.2 Example image (a) from the Sky dataset with its corresponding label (b). The label is the true segmentation mask. A second example image (c) and its label (d) are shown to demonstrate the range of images in the set.

9.2.3 Cell Classification

AL-GP was also applied to cell image data to show how active learning could be applied to the problem of cell segmentation and classification. For this data set, the goal of classification is to determine which of the cells are co-transfected (expressing two proteins of interest). The image data [Ricker et al., 2022] consists of two streams, one that tracks the expression of green fluorescence which indicates the presence of one protein, and another that tracks purple fluorescence, which indicates the expression of the other protein. An example of this data is shown in Figure 9.3.

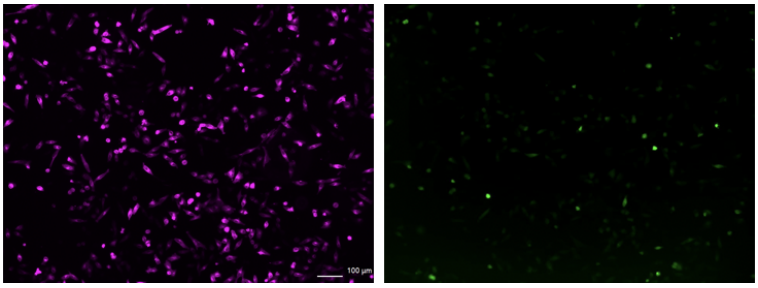


Figure 9.3 The image on the left shows cells that are expressing a protein with a purple fluorescent piece. The image on the right shows cells that are expressing a protein with a green fluorescent piece.

9.3 Active Learning

Active learning was used to iteratively select training samples to maximally inform model populations. Each run begins with a single randomly selected training sample and one additional training sample is selected and added by active learning after a set number of generations. Once the new sample is added, evolution continues on the expanded dataset. Uncertainty was quantified by measuring disagreement between the models in an ensemble. To do this, an ensemble of 10 diverse models is selected from the population. The ensemble of models is then evaluated on every unselected training sample and the uncertainty on each sample is recorded by measuring the average difference between the predictions of each model in the ensemble. The sample with the highest uncertainty value is then selected and added to the training set. This method varies slightly from the original AL-GP approach in that the ensemble sizes are constant and the models are not selected by selecting best fitting models on different data partitions. This change was made since we begin with a single training sample, which would result in one model selected for the first ensemble. A single model lacks any measure for uncertainty.

9.4 AL-GP Applied to Decision Tree GP

Decision tree GP is described in detail in Chapter 7 but is reviewed in some detail here before moving on to the application of AL-GP with DT-GP on several image analysis problems.

9.4.1 Decision Tree GP (DT-GP)

Image segmentation and object classification were performed using an implementation of decision tree GP (DT-GP), where decision trees are evolved to consider which pixels are foreground or background for segmentation and to identify which class an object belongs to for object classification tasks. The implementation of DT-GP is discussed in detail in Chapter 7.

9.4.2 KOMATSUNA Multi-Image Results

DT-GP was tested using the KOMATSUNA dataset to see how active learning impacts its ability to perform on a fairly simple dataset. Active learning was compared to random data selection by recording the test fitnesses after each iteration to see which method improves fitness on the test set

quicker and which method arrives at better fitness values. In this case, a fitness of 1 is a perfect model and a fitness of 0 is the worst possible fitness. Each method was tested a total of 40 times with different randomly selected training and test sets of 250 images available to be selected for the training set and 50 images in the test set.

The results of the 40 runs comparing active learning and random sampling are shown in Figure 9.4. We can see that active learning more quickly improves its test fitness towards 1. We can see though that this is a fairly simple problem for DT-GP since even after just one round of learning the models from both active learning runs and random sampling runs are around 0.98.

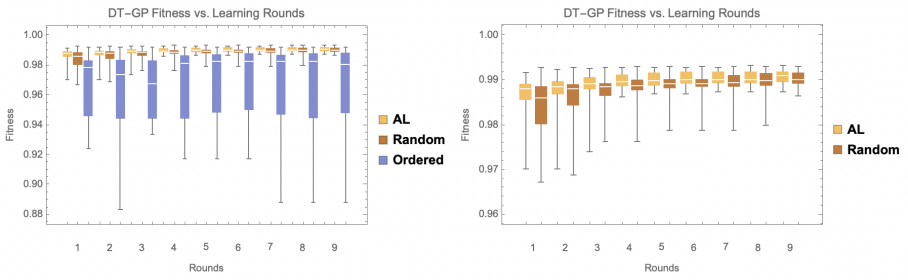


Figure 9.4 Active learning, random data selection, and ordered data selection using DT-GP are compared with the distribution of test fitnesses shown after each round of learning. Each experiment was repeated 40 times, so each bar represents the distributions from 40 repeated trials. The data shows that active learning outperforms random data selection. The active learning method most quickly increases fitness. The right figure focuses on just AL and random data selection for easier visualization of the same results from the left figure.

9.4.3 KOMATSUNA Single-Image Results

DT-GP was also tested using individual images from the KOMATSUNA dataset where instead of using active learning to select full images to add to the training set we begin with a single pixel and add one pixel each round of learning. We compared active learning and random data selection. As before, we record the fitness on a test set after each round of learning. In this case, the test set is the full image and the training set is the subset of pixels selected by the learning strategy. Both active learning and random selection were tested 40 times. The results of the 40 independent trials for one image are shown in Figure 9.5. The results show that active learning outperforms random selection by reducing error more quickly and more consistently, as well, we see that the active learning approach arrives at a lower error. The random selection approach seems to have

gotten stuck in a local optimum since the approach plateaus after around 20 rounds of learning and converges after 30 rounds of learning. The results of a sample run using active learning and random sampling are shown in Figures 9.6 and 9.7. The figures show the original image, the sampled pixels, and the segmentation pattern after 40 points are sampled. The active learning method arrives at a near-perfect segmentation pattern while the random sampling approach results in a segmentation pattern that picks up a lot of background and is missing the center of the plant.

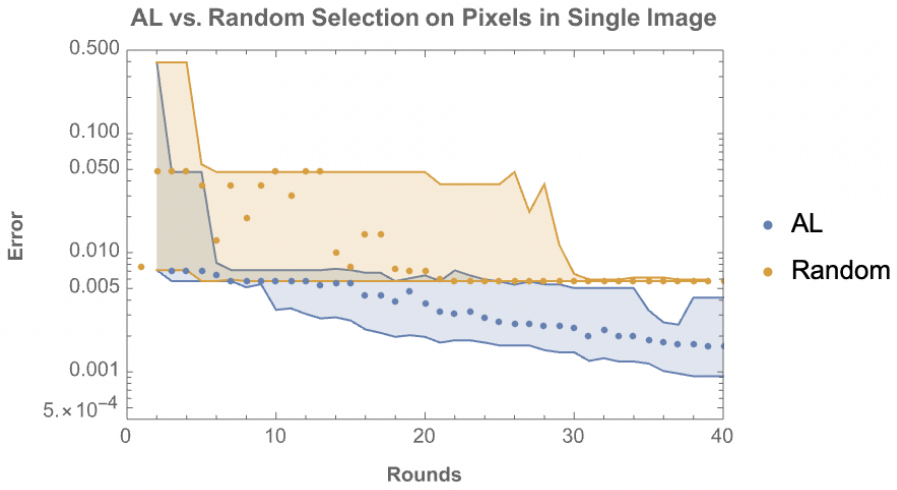


Figure 9.5 Active learning and random data selection using DT-GP are compared on their ability to learn the segmentation of a single image by selecting one pixel each round of learning. Each experiment was repeated 40 times, so the plot shows the median fitness for each method surrounded by bands representing the upper and lower quartiles. The data shows that active learning outperforms random data selection. The active learning method most quickly reduces error and also does not get stuck in a local optimum as the random selection approach does.

9.4.4 Sky Single-Image Results

DT-GP with active learning was tested on single images from the sky dataset as well. We used the same approach for active learning as we did with active learning on individual images from the KOMATSUNA dataset. In Figure 9.8, we can see the result of active learning after 80 iterations compared to the original image and the ground truth segmentation pattern for the sky image '0005'. We can see that the resulting segmentation pattern is close to the ground truth, but there are some regions still being misclassified. Better models could potentially be found with more iterations of active learning. One thing that I found interesting is that there may actually be issues

Figure 9.6 The results of using pixel-based active learning on one of the KOMATSUNA images. The image is shown on the left. In the middle, the 40 sampled pixels are shown, as selected by active learning. On the right, the segmentation pattern is shown, which is nearly perfect.

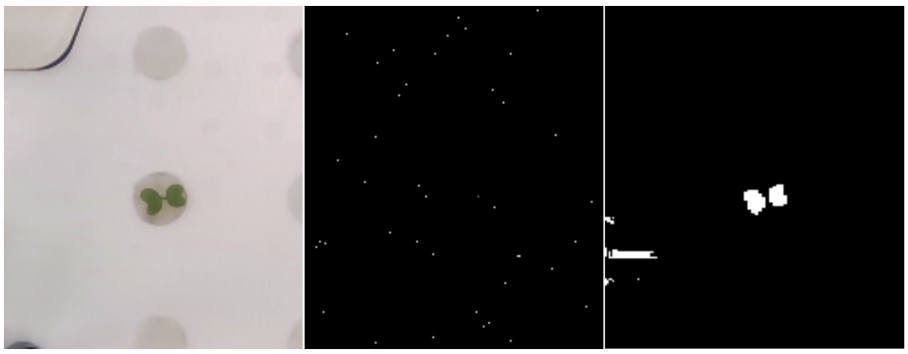


Figure 9.7 The results of using pixel-based random selection on one of the KOMATSUNA images. The image is shown on the left. In the middle, the 40 sampled pixels are shown, as selected using random selection. On the right, the segmentation pattern is shown.

with the ground truth segmentation pattern as is revealed by comparing the segmentation pattern of the ground truth and segmentation pattern of the trained model. If you look at the predicted segmentation pattern on the right just above the roof, you will see a cross shape removed from the sky, this does not get identified by the ground truth segmentation pattern. When looking closely at the original image there appears to be a cross shape that could possibly be a powerline. It seems that this likely should be segmented but isn't in the ground truth for some reason. Having conflicting information in the ground truth segmentation pattern could also provide confusion during training and could lead to suboptimal models.

The training progress over active learning iterations for that same image is shown in Figure 9.9. We can see that after one iteration the developed model does not produce a very good segmentation pattern, but that we get gradual improvement over iterations until the 15th iteration where we find

Figure 9.8 The results of using pixel-based active learning on one of the sky images (image '0005') is shown and compared to the ground truth segmentation pattern and the original image. The original image is shown on the left, the ground truth segmentation pattern in the middle, and the segmentation pattern of the trained model after 80 rounds of AL on the right. We can see that the segmentation pattern is reasonably good, but there are some noticeable problems, meaning either we need more iterations of AL or the image is too challenging to perfectly solve using just the pixel values.

the model that remains the best even after 80 iterations.

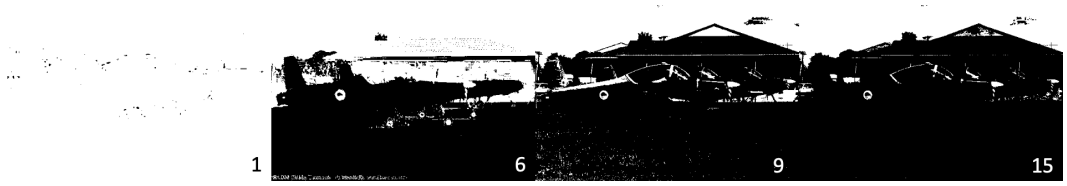


Figure 9.9 The results of using pixel-based active learning on one of the sky images (image '0005') over several iterations of AL is shown. The segmentation after 1 iteration is shown on the left, with subsequent segmentation patterns shown to the right with their iteration number indicated in the bottom right corner. We can see that initially, the segmentation pattern is very bad and over iterations it improves drastically.

The progress over iterations of the 40 repeated trials is compared between both random and active learning and shown in Figure 9.10. We can see that the median error when using AL ends up lower than when using random selection after 80 iterations. We can also see that AL arrives at lower errors sooner than random selection, this is most notable around 40 iterations.

In Figure 9.11 we can see the learning progress when using active learning on sky image '0008'. The image after the first segmentation pattern after the first iterations is all one color, so the figure starts with the second iteration. We can see the gradual improvement over iterations. It is interesting to notice that the text label in the original image does cause some confusion, as shown in the second segmentation pattern. This is an issue of poor-quality data since labels should not be in the raw images. The models do learn how to overcome that issue though, and no longer get confused with

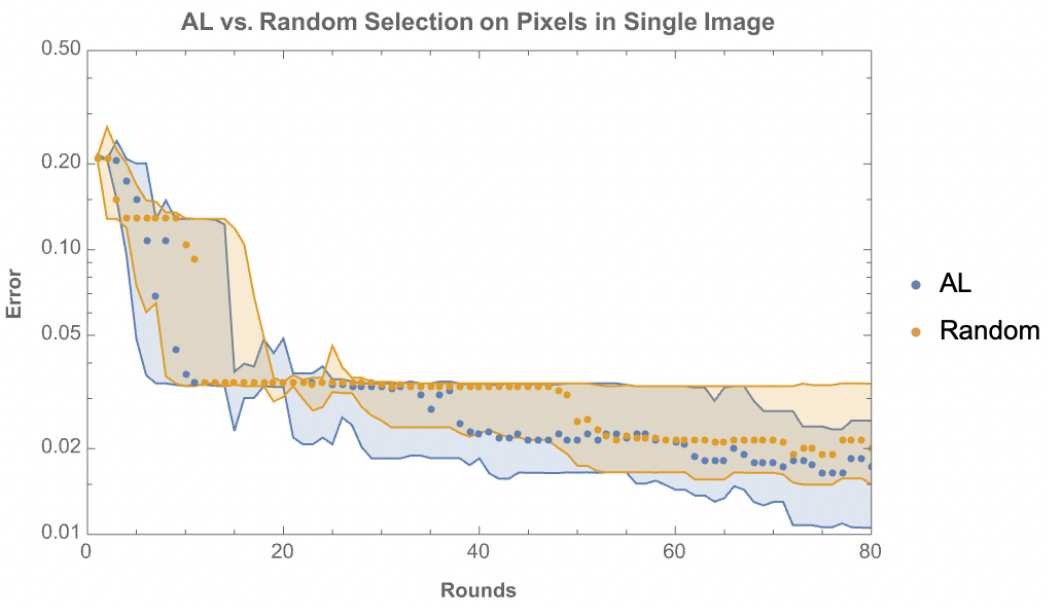


Figure 9.10 The results of comparing AL and random selection are compared for the sky image '0005'. We can see that the AL approach ends up at a lower median error after 80 iterations. As well, we can see that the AL approach reaches lower errors sooner than random, although random does eventually catch up.

the labels in later iterations.

Figure 9.11 The results of using pixel-based active learning on one of the sky images (image '0008') over several iterations of AL is shown. The segmentation after 2 iterations is shown on the left, with subsequent segmentation patterns shown to the right with their iteration number indicated in the bottom right corner. We can see how the segmentation pattern improves over iterations.

The corresponding models for each segmentation pattern in Figure 9.11 are shown in the same order in Figure 9.12. It is interesting to note that the models do not exclusively get more complex. They initially increase in complexity before finding the simple yet accurate model corresponding to the 36th iteration. Once this simple yet accurate model was found, the models increased in complexity until arriving at the final, fairly complex, model. It is interesting to note that the model from the 36th iteration seems to be used as a branch in the final model, with one change, that the "LessEqual" has been changed to "Greater", making it equivalent to the application of the *Not*

operator on the earlier model. Looking at the final model does seem to indicate that DT-GP could benefit from a selection strategy that promotes simpler models since there are two branches that do not provide any function since they always evaluate to false.

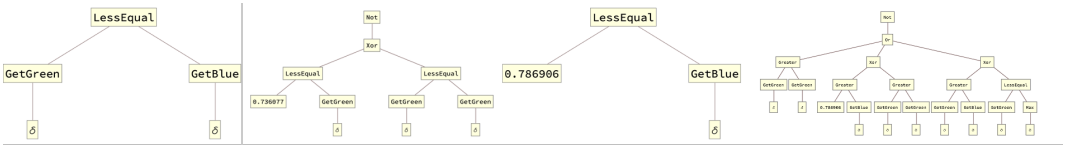


Figure 9.12 The corresponding models to the segmentation patterns shown in Figure 9.11 are displayed here.

Figure 9.13 shows a comparison between the distributions of errors when using random selection and active learning over the 40 repeated trials. We can see that active learning arrives at lower errors after 80 iterations. We can also see that around 60 iterations the active learning approach decreases in error more quickly than random selection.

Figure 9.13 The results of comparing AL and random selection are compared for the sky image '0008'. We can see that the AL approach ends up at a lower median error after 80 iterations. As well, we can see that the AL approach reaches lower errors sooner than random, most noticeable around round 60.

A segmentation pattern from one of the AL runs is compared to the original image and the ground truth segmentation pattern in Figure 9.14. We can see that the evolved model creates a

reasonably good segmentation pattern, but is still not perfect. There are issues though in the ground truth that could make it more challenging to arrive at a perfect segmentation pattern since even the ground truth is not really "truth". At least one noticeable issue exists in the ground truth, and that is that there is clearly some sky visible under the tail of the plane, yet the ground truth does not consider it to be sky.

Figure 9.14 The results of using pixel-based active learning on one of the sky images (image '0008') is shown and compared to the ground truth segmentation pattern and the original image. The original image is shown on the left, the ground truth segmentation pattern in the middle, and the segmentation pattern of the trained model after 80 rounds of AL on the right. We can see that the segmentation pattern is reasonably good, but there are some noticeable problems. There are also issues with the ground truth since there is clearly some sky appearing under the tail of the plane, but it is not identified in the ground truth.

9.4.5 Cell Classification

We applied DT-GP to automate the process of identifying cells in videos and determining which cells are co-transfected to help accelerate a research project where progress is slowed by time spent manually labelling every cell in each video frame. Co-transfected cells are those that express two different proteins of interest that the researcher instructed the cell to produce by inserting the genetic information into the cells. To identify co-transfected calls, the cells have to be observed under two different light filters. This leads to two sets of images that have to be considered to identify which cells contain both proteins. Each filter will reveal which cells contain one of the proteins. The task of labelling cells in images is currently done manually and occupies a significant amount of time since it requires cells to be outlined in each video frame as well as requires analysis of the second set of images to determine which of the cells in the video stream are co-transfected. This makes it an ideal task to be automated since automation could lead to significant time savings and enhance research productivity.

The goal of this project is to utilize a GP system with an active learning strategy to develop models that can automate the process of identifying cells and labelling them as co-transfected or not. Two types of models are developed. The first being a model that can correctly identify all the viable cells in the image. The second type of model identifies which of the cells are co-transfected. The goal of applying active learning is to only require a few cells be labelled and then the developed models would be able to correctly select and classify the rest of the cells in that frame as well as additional frames if needed. Once all of the cells are labelled the researcher can focus on the analysis of the data sooner.

The first stage of modeling classifies pixels as either being part of a cell or being background. The second stage then identifies if a cell is co-transfected or not by analyzing the two different images from different filters. Using the results from identifying which points are part of a cell or not, a clustering algorithm is used to associate connected points and identify them as individual cells. An active learning strategy was implemented to guide labelling of the cells. This is done by presenting the user with points or cells that have maximal disagreement between their classifications amongst the models in an ensemble. The user can then provide a true label for the cell or point. That cell or point with the label is then added to the training set.

Figure 9.15 shows some results where a developed model was tasked with identifying all the co-transfected cells in an image. The two images were overlayed so we can observe both the green and purple colors. It seems that generally, the purple color is more intense so it dominates most of the images. The initial segmentation is done on the green images, so despite the images appearing purple, green is in fact present in all of the cells displayed.

Looking closely at how the points are grouped together as single cells in Figure 9.16 indicates that improvements could be made in how the clustering is done. There are regions where it is clear that two separate cells should be identified, yet they are grouped together as a single cell. Rather than just connecting all adjacent pixels that are identified as cell material, it may be necessary to fine-tune the method to look for indicators of a cell membrane to separate nearby cells.

One of the cell classification models found is shown in Figure 9.17. This model was able

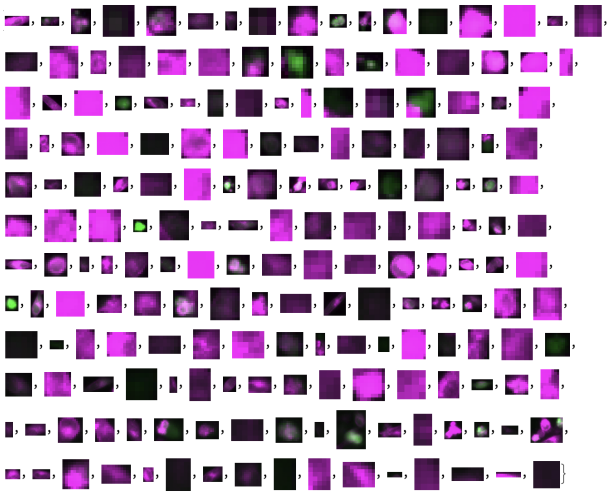


Figure 9.15 These are the cells identified as co-transfected by the model developed with the GP system. Each box is supposed to fully encapsulate a co-transfected cell. Both the green and purple images have been combined here to show how both colors are present in the images.

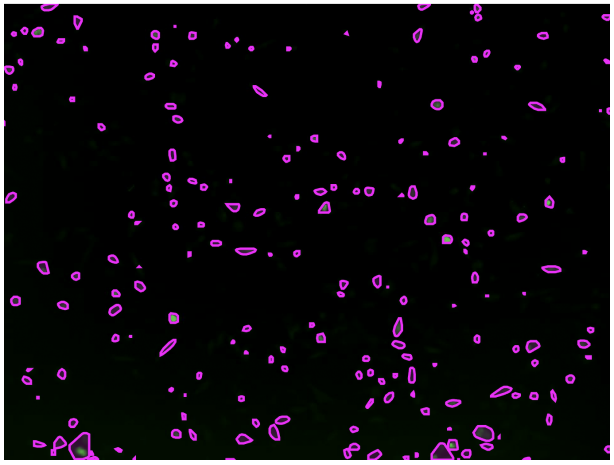


Figure 9.16 Shown here are the outlines of each cell that result from the model selecting points that are either cell or non-cell material and connecting the adjacent ones. Looking near the bottom left corner we can see an example of where two cells are incorrectly identified as a single cell.

to correctly identify all the cells in a test set. The model can be interpreted as follows: If the max blue value minus the min blue value is greater than the average green value, then the cell is co-transfected, otherwise, it is not co-transfected. This interpretation ignores the left half of the tree since the left half of the tree reduces to a constant value of false.

To make the use of AL-GP with DT-GP easy to use for this project, a simple GUI was developed that allows the user to fine-tune a pre-trained segmentation model and then supply labels for cells identified as uncertain by the classification models. A snapshot of the GUI in use is shown in

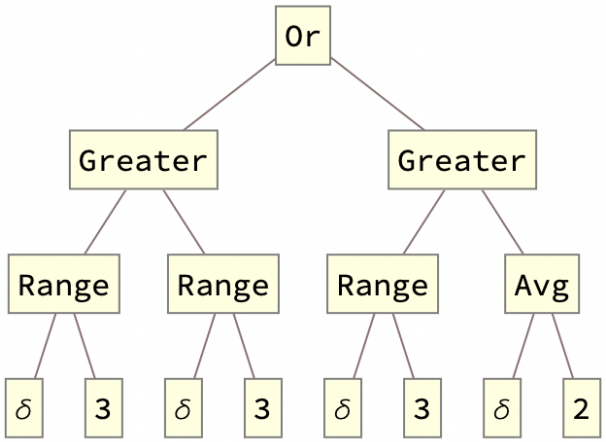


Figure 9.17 Here is a graphical representation of a model's genotype that was evolved and able to correctly classify a test set. In the tree, "Range" is a function of the max value minus the min value in a specific color stream of the data. The data is stored with three values for each pixel, the red, green, and blue values in that order. So "Range" with the value 3, means the max minus the min value in the cell in the blue values. "Avg" is a function of the average value in a specific color. Looking at this tree we can see that the left side of the tree would always evaluate to false, so only the right side of the tree contains the effective code.

Figure 9.18. The use of this GUI allowed for automation of cell classification after labelling just a few cells, where previously it was necessary to manually label every cell in every video frame.

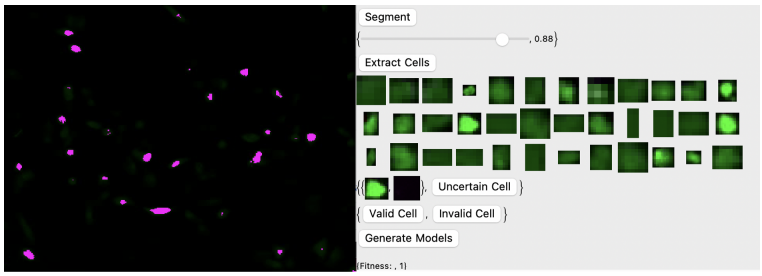


Figure 9.18 The image shows the GUI developed to make easy use of AL-GP to develop models to classify cells. The slider allows the user to fine-tune a pre-trained segmentation model. The segmented cells are then shown to the user and when "Uncertain Cell" is clicked it presents the two image streams for the cell identified as most uncertain given the current classification models. The user can then select if the cell is co-transfected or not by selecting "Valid" or "Invalid". Once a selection is made, the user can then click "Generate Models" to continue model development on the expanded training set. The fitness on the training set is displayed at the bottom to give the user a sense of the model quality on the training set. Once a few cells have been labelled by the user and a model of sufficient quality is achieved the model can then be deployed and applied to the remaining cells in each video frame.

Parameter	Setting
Mutation Rate	90
Crossover Rate	90
Spawn Rate	100-HoF-UniqueMods
Hall of Fame Size	10
Tournament Size	4
Population Size	100
Generations per AL Iteration	100

Table 9.1 SEE-Segment Parameter Settings.

9.5 AL-GP Applied to SEE-Segment

9.5.1 SEE-Segment

SEE-Segment was used to evolve a population of image segmenters. Each individual model consists of a segmentation strategy and 8 parameters. The parameters and strategy are optimized through generations. While each individual contains 8 parameters not all parameters are used by all strategies. Mutation and crossover are used as the variation operators. Tournament selection is utilized for selection. All models selected from tournament selection then have a 90% chance of crossover and then from those offspring, there is a 90% chance of a mutation. Elitism is also utilized by guaranteeing the hall of fame models (top 10 genotypically unique models) are preserved across generations. The parameters used to run SEE-Segment are shown in Table 9.1. Spawn rate refers to the number of new models introduced in a generation. The number spawned each generation is the number required to bring the population size back up to 100 after including the hall of fame models and the unique individuals produced from mutation and crossover. Tournament size, population size, and generations are the only parameters modified from the default settings. Those settings were chosen since they allowed for sufficient model development while keeping computation time reasonable in each iteration of learning.

9.5.2 AL Implementation for SEE-Segment

Two active learning methods were implemented that vary in how they select the ensemble. The first method simply uses SEE-Segment's hall of fame, which consists of the top 10 genotypically unique models. The second method goes further to ensure models are unique by selecting the

top 10 phenotypically unique models. These ensemble selection methods differ from the original AL-GP approach since here we start with only a single image. The original approach relies on generating diverse data clusters and selecting models that best fit each data cluster. If that approach was utilized here, we would initially get an ensemble of one model, which would lack any sort of uncertainty metric.

Using the selected ensemble we compute the average pairwise disagreement of all the models in the ensemble on each potential image. The image with the maximum average pairwise disagreement is selected for labelling and added to the training set. In the event of a tie, the first image found with the max value is selected. An overview of the active learning algorithm is shown in Algorithm 9.1 and the method for computing uncertainty is shown in Algorithm 9.2. The pairwise uncertainty on an image between two models is computed by utilizing SEE-Segment's fitness function where instead of supplying the fitness function with a model's predicted segmentation mask and a true mask, the predicted masks from both models are supplied. This leads to a reasonable measure of how different two model's predicted masks are.

```
TrainingData ← RandomImage                                  ▷ Select 1 Random Image
Evolver ← GeneticSearch.Evolver(TrainingData)               ▷ Initialize evolver
evolver.run()                                               ▷ Evolve models with initial data
while i ≤ iterations do                                     ▷ While max iterations not reached
    HoF ← Evolver.hof                                       ▷ Extract hall of fame
    SelectedImage ← MaximizeUncertainty(HoF, Data)          ▷ Find image that maximizes uncertainty
    TrainingData ← Append(TrainingData, SelectedImage)      ▷ Add new image to training data
    Evolver.UpdateData(TrainingData)                        ▷ Update training data in evolver
    evolver.run()                                           ▷ Evolve models with new data
end while
```

Algorithm 9.1 Active Learning Process for SEE-Segment.

```
procedure UNCERTAINTY(HoF,Image)
    Uncertainties ← []                                      ▷ Initialize uncertainties list
    Pairs ← GeneratePairs(HoF)                              ▷ Generates all pairs of models in HoF
    for i 1 to len(Pairs) do
        Prediction1 ← EvaluateModel(Pairs[i][0], Image)     ▷ Generate prediction of first model in pair
        Prediction2 ← EvaluateModel(Pairs[i][1], Image)     ▷ Generate prediction of second model in pair
        UncertaintyVal ← FitnessFunction(Prediction1, Prediction2)  ▷ Compute uncertainty
        Uncertainties ← Uncertainties.append(UncertaintyVal)        ▷ Append to list
    end for
    MeanUncertainty ← mean(Uncertainties)                   ▷ Compute mean of uncertainties
    Return MeanUncertainty
end procedure
```

Algorithm 9.2 Uncertainty Computation SEE-Segment.

9.5.3 KOMATSUNA Results

To determine the success of active learning on the KOMATSUNA data set, active learning was compared to two naïve image selection methods. The first method is ordered selection of training data, where the images are added to the training set in their natural order in the data set. Specifically, images in the KOMATSUNA data set are from a time series and labelled in order, so the order would be from first to last in the time series. The second method is random order selection, where a new image is added to the training set randomly.

The KOMATSUNA data set contains 300 images. Of the 300 images, 50 of them were reserved as the test set and the remaining 250 were available to be selected for training. Each approach began with a single image in the training set and could select one new image to be added to the training set each iteration.

The results of comparing active learning, random selection, and ordered selection are shown in Figure 9.19. The results show that ordered selection performs very poorly, having the worst fitness values on average and also having the widest distribution of fitness values. This is not surprising since the data set is ordered as a set of several time series. Images nearby in the sequence will be very similar since they are of the same plant within a short period of time. This makes it challenging for models trained only using the beginning of the time series to predict the segmentation patterns correctly of plants that are much larger later in the time series. Random sampling and active learning both eventually achieve similar test errors as seen in Figure 9.19, but active learning achieves low error in fewer iterations and converges more quickly. The ability of random sampling to achieve good fitness in relatively few iterations is likely a result of the data set being very balanced, so a random sample that is large enough will contain data representative of the whole set. The key difference between active learning and random sampling is the rate at which low test error is achieved and the consistency of low error solutions in few rounds of learning.

To determine how well active learning can overcome unbalanced data, the KOMATSUNA data set was modified to heavily oversample 3 images by duplicating 3 random images 50 times and adding them back into the training set. The test set is still balanced since it was separated prior

to the duplication of the images. The results of comparing the two active learning methods and random data selection are shown in Figure 9.20. Ordered sampling was not included here since it already performed poorly with the balanced data. The results show that already after the second iteration, both active learning approaches are beginning to converge to a good solution. By the third iteration the AL (Div) approach appears to have converged, and the AL (HoF) approach appears to converge by the 5th iteration. After the 6th round, random still has a fairly wide distribution compared to the active learning approaches.

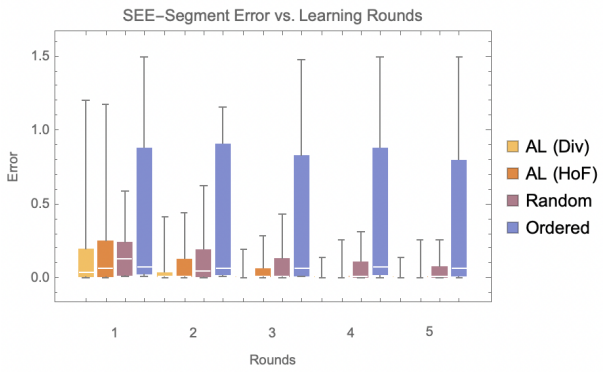


Figure 9.19 Two active learning methods are compared with ordered data selection and random data selection with the distribution of errors shown after each round of learning. Each experiment was repeated 40 times, so each bar represents the distributions from 40 repeated trials. The data shows that active learning outperforms both random data selection and ordered data selection. The active learning methods most quickly decrease error and then most quickly converge. On and after round 5 we see that from the minimum to the 3rd quartile the active learning error distribution appears essentially as a flat line. AL (Div) represents AL when using a diverse ensemble and AL (HoF) represents AL when using the HoF as the ensemble. We see that using a diverse ensemble improves performance.

Figures 9.21 and 9.22 compare a sample run of active learning and ordered selection by tracking the segmentation pattern on a sample from their test sets. Figure 9.21 demonstrates how active learning develops a good model quickly and improves over iterations. Comparing this to Figure 9.22, we see how there is risk when using suboptimal sampling strategies to overfit training data and perform poorly on test data. We see that the ordered selection method actually gets worse by the fifth iteration, indicating that it is overfitting a non-representative training set. Even by the 20th iteration ordered selection arrives at a rather poor model.

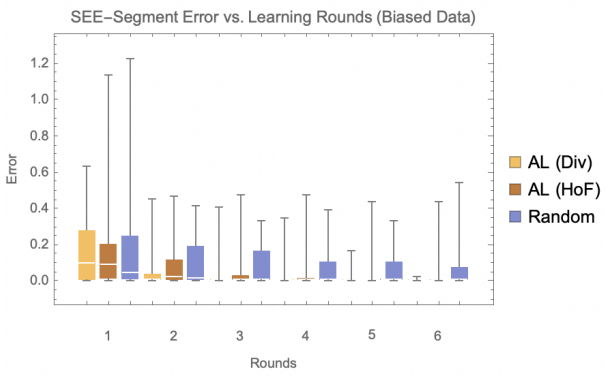


Figure 9.20 Two active learning methods are compared with random data selection on the biased KOMATSUNA data. The distribution of errors is shown after each round of learning. Each experiment was repeated 40 times, so each bar represents the distributions from 40 repeated trials. The data shows that active learning outperforms random data selection. We also see that again the active learning method using diverse ensembles performs better than the method using the hall of fame.

9.6 Conclusions

AL-GP was extended to two new machine learning systems, DT-GP and SEE-Segment, for the purpose of accelerating the development of image processing models. It was shown that AL-GP can successfully be extended to tasks outside of symbolic regression and also to other population-based machine learning systems. When applying AL-GP to DT-GP we verified that active learning accelerates model development on the KOMATSUNA dataset, which is a simple dataset to model for DT-GP. Once verifying AL-GP is applicable to DT-GP we applied it to the task of cell classification to aid another lab at MSU in accelerating their research by partially automating the task of classifying cells by using a human-in-the-loop method where the human researcher supplies labels for the cells identified as most informative by the GP system. This results in just a few samples requiring human labelling before finding models that could then be deployed to automatically label the rest of the cells.

When applying AL-GP to SEE-Segment we explored how active learning compares to random sampling and ordered sampling. We also explored how well it can overcome biased data and compared two different methods for selecting ensembles to be used in the uncertainty computation. We observed that both active learning methods outperformed random and ordered sampling by finding better solutions more quickly and consistently across 40 repeated trials. We also observed

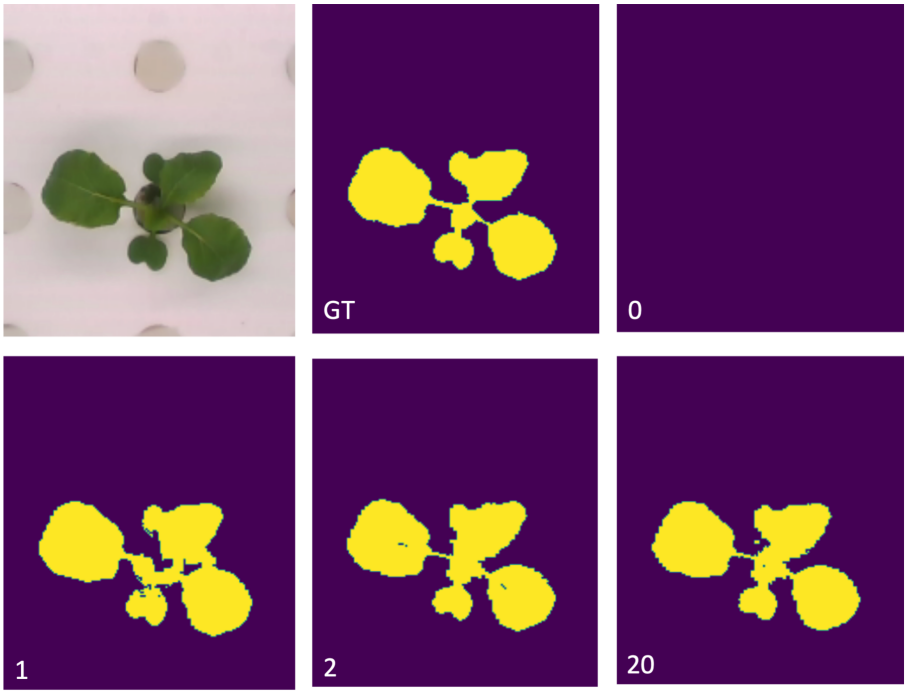


Figure 9.21 Shown here is the progress from an example run using active learning on the KOMATSUNA plant data. The top left image shows the original image and the top middle shows the ground truth segmentation pattern. The top right begins the active learning iterations and continues down then to the right. After the initial random image, we see that the best model is terrible on an image from the test set. We see quick improvement and convergence to a good solution. The segmentation patterns shown are from the 0th, 1st, 2nd, and 20th iterations of active learning, where the 0th represents training on the first randomly selected image used to seed the training set.

that biasing the data did not have a significant impact on the active learning approaches, still outperforming random sampling in finding good solutions more quickly and consistently. The ensemble selection method that used phenotypic diversity was found to perform better than the method that used genotypic diversity. This shows that a diverse ensemble is an essential part of the AL-GP approach.

Additional work is currently underway exploring how AL-GP impacts SEE-Segment's model development on a more challenging dataset where near-perfect solutions are not achievable. The preliminary results indicate that AL is developing better models with fewer training samples, but more runs are still needed to confirm these results.

This work confirms that AL-GP can be successfully applied to population-based ML systems outside of StackGP and further that AL-GP is not restricted to regression tasks. It also shows

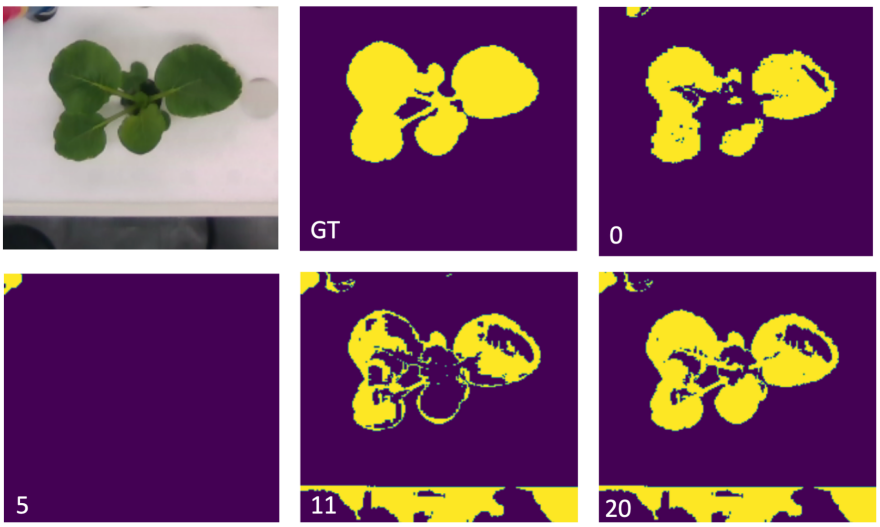


Figure 9.22 Shown here is the progress from an example run using ordered selection on the KOMATSUNA plant data. The top left image shows the original image and the top middle shows the ground truth (GT) segmentation pattern. The top right begins the ordered selection iterations and continues down then to the right. After the initial random image, we see that the best model is actually not too bad. By the 5th iteration, we actually see that the performance on this example test image actually worsens significantly. By the 20th iteration, we see some improvement but the model is still performing poorly.

how AL-GP can be applied to help accelerate research projects by reducing the time and cost of collecting and labelling training samples.

Acknowledgement: Computer support by MSU's iCER high-performance computing center is gratefully acknowledged.

CHAPTER 10

FUTURE WORK

It was realized that one of the reasons that correlation is a good fitness function for symbolic regression tasks is that correlation is a *relative* fitness function with the additional feature that correlation allows for a shortcut (linear regression) to be applied to a good solution to align the solution. This idea seemed extendable to classification tasks. For binary classification tasks, the extension is obvious, just remap the outputs to the opposite label for any model that has a classification error of more than 50%. To extend this to N-class problems I had the idea that a fitness function could leverage the confusion matrix. The fitness function would choose one value from each row of the confusion matrix, such that the position of each value in each row is unique, and also in a way such that the total is maximized. By doing this, you are then able to use the row positions that lead to the maximal fitness value to remap the model's predicted labels to correct labels. This fitness function has the potential to be expensive, so maybe this fitness function is only used every N generations and in between a different fitness function is applied that doesn't search for a best mapping and just uses a mapping inherited from the parent. Future work to explore if this fitness function leads to improved performance on GP classification tasks is planned.

The results of trying two different ensemble selection methods when using SEE-Segment indicated that the diversity of the ensemble has a significant impact on the success of active learning. This reveals that further work should be done to explore how an ensemble should be selected to capture the diversity of a population in a way that leads to the best performance of active learning. Various ensemble selection methods could be explored for each problem domain such as regression, classification, and image segmentation.

It also seems that the methods developed for active learning could be utilized in data subsampling methods, where in every few generations an active learning-inspired approach could be used to subsample the training set and just use the maximally informative subset for those generations. It could potentially improve the speed of evolution by only requiring that informative samples be included in the fitness evaluations. Potentially it could also improve the quality of models if bias in

the data is minimized by sampling the diversity of the data.

In all of our active learning methods, we began with a randomly selected initial set. It could be possible to use an unsupervised active learning strategy to select representative samples so that even the initial set contains a relatively high amount of information. It is possible that our strategy could be more efficient if starting off with high-quality training samples.

For the image segmentation problems, I explored two types of active learning, an image-based approach and a pixel-based approach. The image-based approach requires relatively few samples to be selected since each sample has a lot of information, but when training on whole images, model development becomes expensive. Alternatively, pixel-based approaches are very fast to evaluate since it is quick to evaluate fitness on a small set of pixels, but unfortunately, for a complex problem, the number of iterations of active learning quickly grows to be unreasonable. I'm currently considering developing an image subsampling approach that just selects regions within images to add to the training set. This could strike a better balance between having samples that are informative, while also keeping the cost to evolve models reasonable. I'm considering two approaches for this now. A fixed sample size approach that partitions each image into fixed-size sub-images and the active learning method scans over those smaller images. The second approach I'm considering would be more flexible and would use an unsupervised method, such as clustering, to identify different regions within images, where those regions would then be considered for selection by active learning. The development of this approach will be essential to succeeding in the new collaboration with the Toulouse, France research group, where we are trying to use active learning to help develop Cartesian GP models that can identify tumors in medical imaging data.

CHAPTER 11

CONCLUSIONS

Active learning is a widely applied method within machine learning to focus model training on samples that are most informative, thus minimizing costs associated with collecting and labelling data, while accelerating model development. In this dissertation, I take inspiration from active learning in other fields and develop approaches to apply active learning to population-based machine learning systems on a range of problem domains. In pursuit of developing active learning strategies for population-based machine learning systems, I developed several new genetic programming systems and also identified benefits of using relative fitness measures, such as correlation, to improve visibility of solutions in the search space.

In Chapter 4, I describe the implementation of my StackGP system. It is a dual-implemented system in both Mathematica and Python. This system is unique in that it uses a stack-based model representation similar to PushGP, as opposed to the more common tree-based GP. The system's performance is also attributed to the use of Pareto tournament selection, which considers both simplicity and accuracy simultaneously when selecting models, and the use of correlation as the accuracy metric with a final alignment step to adjust the parameters of models selected using the correlation measure. For active learning in population-based machine learning, a model ensemble is key for determining uncertainty, so I introduce a method for selecting a model ensemble that captures high-quality yet diverse models by clustering the training data and selecting models from the population that best fit each data cluster, ensuring no model is selected more than once, and also defaulting to using the Pareto front of the population in the event that fewer than three data clusters were created by the clustering method. Having a high-quality GP system is essential for doing active learning since the GP system has to be capable of extracting the information from the training samples selected by active learning.

In Chapter 5, I compare RMSE and correlation when used as fitness measures for GP symbolic regression tasks. I demonstrate that correlation when used with a simple alignment step significantly outperforms the traditionally used RMSE on a wide range of benchmark problems. It

was demonstrated that correlation performs better with fewer data points and also performs better when the data is corrupted with low levels of noise. This work was presented at GPTP 2022 and published here [Haut et al., 2023a]. This chapter serves as a discussion on why within the field of GP we should be considering ways to use relative fitness measures that improve solution visibility within the search space and shrink the search space. When using correlation as the fitness measure all solutions of the form $y = a * f(x) + b$ for any a or b become visible and the search space is reduced since the specific values for a and b do not have to be discovered by the GP search and are rather found by linear least squares regression, which is much better suited for that specific task.

Chapter 6 explores how various uncertainty and diversity measures can be used to guide active learning for GP symbolic regression tasks on their own and together using a Pareto optimization strategy. Uncertainty methods consider a model ensemble selected from the population to find new data points where there is maximal disagreement. Diversity methods consider data points that are most unique relative to the already selected training points. Of the uncertainty metrics explored, we show that the differential entropy method performs best. The results of comparing the uncertainty functions were presented at GECCO 2023 and are published in the proceedings here [Haut et al., 2023b]. When comparing data diversity metrics, we show that lack of correlation between data points performs best, but has the drawback that each data point must have at least 3 dimensions. Correlation between data points is determined by evaluating the correlation between each point as a vector of their values in each dimension. The Pareto optimization approach considers both data diversity and uncertainty when selecting new points, with the goal of selecting new points that have a good balance between diversity and uncertainty. This helps reduce the risk of repeatedly selecting similar points which can occasionally happen when using just uncertainty, while also ensuring the selected points will be informative to the current model population. It was shown that the multi-objective approach often improves the stability of active learning and can occasionally lead to performance gains when compared to either diversity or uncertainty on their own.

The second half of the dissertation focuses on classification and image analysis problems instead of regression problems. To address those types of problems I developed the DT-GP system, which

is described in Chapter 7. DT-GP uses a unique hierarchically typed model structure that ensures every developed model is a valid decision tree. This leads to an efficient search since no time is wasted on invalid models. This system also benefits from the fact that it utilizes arbitrary arity since boolean operators are not functionally restricted to operating on just two inputs. This allows the system greater flexibility since rather complex expressions can be discovered using simple representations that are more easily reachable in an arbitrary arity setting. This system also utilizes a relative fitness measure with an alignment step, analogous to correlation for regression tasks. This fitness measure considers each model as both the raw model and its exact opposite during evolution and then assigns the correct state during the alignment when model development is terminated. For DT-GP, a simpler method for generating ensembles was used, which is to simply select the top 10 models from the population that are phenotypically unique.

Active learning was also explored using SEE-Segment, which is a population-based ML system that was specifically designed to evolve image segmentation algorithms. I described this system in Chapter 8, focusing on an overview of the system and describing in detail my specific contributions to this system. Specifically, I improved the search strategy by improving its diversity management by weakening selection pressure by instead using tournament selection with small tournament sizes, instead of strictly using an elitist approach that selects only the top N models in each population globally for selection. I also introduced caps on how many duplicates can exist in any population. I set the cap to 1 for my runs, but made it adjustable so more duplicates can be allowed. I also developed an uncertainty-based active learning strategy with two different model ensemble options. The first option uses the built-in Hall of Fame feature which contains the top 10 genotypically unique models, while the second options is a bit stricter in diversity and selects the top ten phenotypically unique models.

The active learning implementations for both DT-GP and SEE-Segment were evaluated in Chapter 9 on several different data sets. DT-GP was first evaluated on its ability to select maximally informative images from a set of images to reduce the number of images required to find a global solution that can segment a class of images. This was shown to be successful compared to random

selection of images when using the plant KOMATSUNA dataset. DT-GP was then evaluated on its ability to select maximally informative pixels within individual images to find a model that fits the rest of the image using as few pixels as possible. This was shown to outperform random selection on both the KOMATSUNA and sky datasets. It was then demonstrated how DT-GP was applied in a real research setting to accelerate and automate the task of segmenting and classifying biological cells in video data. Active learning with SEE-Segment was applied to the KOMATSUNA dataset and it was shown to outperform random and ordered selection in regard to the number of images required before finding quality segmentation algorithms. It was also shown that the stricter approach for considering model diversity when selecting the model ensemble. This approach was the one that selected the phenotypically unique models.

Overall, I demonstrated active learning with genetic programming to be a versatile approach that can be applied to a wide range of population-based machine learning systems across several diverse problem domains. The results in this dissertation show that active learning can be used to reduce training data requirements needed to achieve high-quality models. This makes it an ideal method to apply to any research setting where data acquisition or data annotation is expensive or time-consuming since active learning can reduce the number of samples required. I also hope this work will inspire future research into how we can further improve active learning methods to better inform models through evolution and also develop new understandings into how the choice of training samples influences the development and path of evolution.

BIBLIOGRAPHY

Eduardo B. Alexandre and Paulo A.V. Miranda. Sky Dataset. URL https://www.ime.usp.br/~eduardob/datasets/sky/.

T. Back and H.-P. Schwefel. Evolutionary computation: an overview. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 20–29, 1996. doi: 10.1109/ICEC.1996.542329.

James Edward Baker. Adaptive selection methods for genetic algorithms. In *Proceedings of the first international conference on genetic algorithms and their applications*, pages 101–106. Psychology Press, 2014.

Wolfgang Banzhaf, Peter Nordin, Robert Keller, and Frank Francone. *Genetic Programming — An Introduction.* Morgan Kaufmann, San Francisco, CA., 1998.

Alberto Bartoli, Andrea De Lorenzo, Eric Medvet, and Fabiano Tarlao. Active learning of regular expressions for entity extraction. *IEEE Transactions on Cybernetics*, 48(3):1067–1080, 2018. doi: 10.1109/TCYB.2017.2680466.

Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies –a comprehensive introduction. *Natural Computing*, 1(1):3–52, 2002.

Markus F. Brameier and Wolfgang Banzhaf. *Linear Genetic Programming*. Springer, New York, NY., 2007.

David A. Cohn, Zoubin Ghahramani, and Michael I. Jordan. Active learning with statistical models. *Journal of Artificial Intelligence Research*, 4(1):129–145, mar 1996. ISSN 1076-9757.

Robert Curry, Peter Lichodzijewski, and Malcolm I Heywood. Scaling genetic programming to large datasets using hierarchical dynamic subset selection. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(4):1065–1073, 2007. doi: 10.1109/TSMCB.2007.896406.

Junio De Freitas, Gisele L Pappa, Altigran S da Silva, Marcos A Goncales, Edleno Moura, Adriano Veloso, Alberto HF Laender, and Moisés G de Carvalho. Active learning genetic programming for record deduplication. In *IEEE Congress on Evolutionary Computation*, pages 1–8, 2010. doi: 10.1109/CEC.2010.5586104.

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002. doi: 10.1109/4235.996017.

Finale Doshi-Velez, Joelle Pineau, and Nicholas Roy. Reinforcement learning with limited reinforcement: Using bayes risk for active learning in pomdps. *Artificial Intelligence*, 187-188: 115–132, 2012. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.2012.04.006. URL

https://www.sciencedirect.com/science/article/pii/S0004370212000458.

Ibrahim M. El-Hasnony, Omar M. Elzeki, Ali Alshehri, and Hanaa Salem. Multi-label active learning-based machine learning model for heart disease prediction. *Sensors*, 22(3), 2022.

Evolved_Analytics. DataModeler. URL https://evolved-analytics.com/datamodeler-app/.

Natalie Eyke, William Green, and Klavs. Jensen. Iterative experimental design based on active machine learning reduces the experimental burden associated with reaction screening. *Reaction Chemistry & Engineering.*, 5:1963–1972, 2020.

R. Feynman, R. Leighton, and M. Sands. *The Feynman Lectures on Physics*, volume 1. Basic Books, New York, NY, 1963a.

R. Feynman, R. Leighton, and M. Sands. *The Feynman Lectures on Physics*, volume 2. Addison Wesley, Boston, MA, 1963b.

R. Feynman, R. Leighton, and M. Sands. *The Feynman Lectures on Physics*, volume 3. Addison Wesley, Boston, MA, 1963c.

Gary B. Fogel. *Evolutionary Programming*, pages 699–708. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 1989.

Nathan Haut, Wolfgang Banzhaf, and Bill Punch. Active learning improves performance on symbolic regression tasks in stackgp. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '22, page 550–553, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392686.

Nathan Haut, Wolfgang Banzhaf, and Bill Punch. *Correlation Versus RMSE Loss Functions in Symbolic Regression Tasks*, pages 31–55. Springer Nature Singapore, Singapore, 2023a. ISBN 978-981-19-8460-0. doi: 10.1007/978-981-19-8460-0_2. URL https://doi.org/10.1007/978-981-19-8460-0_2.

Nathan Haut, Bill Punch, and Wolfgang Banzhaf. Active learning informs symbolic regression model development in genetic programming. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, GECCO '23 Companion, page 587–590, New York, NY, USA, 2023b. Association for Computing Machinery. ISBN 9798400701207. doi: 10.1145/3583133.3590577. URL https://doi.org/10.1145/3583133.3590577.

Tin Kam Ho. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 1, pages 278–282 vol.1, 1995. doi: 10.1109/ICDAR.1995.598994.

John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992a.

John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992b.

Maarten Keijzer. Improving symbolic regression with interval arithmetic and linear scaling. In *European Conference on Genetic Programming*, pages 70–82. Springer, 2003.

Michael F Korns. Large-scale, time-constrained symbolic regression. In *Genetic Programming Theory and Practice IV*, pages 299–314. Springer, 2007.

Michael F Korns. Large-scale, time-constrained symbolic regression-classification. In *Genetic Programming Theory and Practice V*, pages 53–68. Springer, 2008.

Michael F Korns. Symbolic regression of conditional target expressions. In *Genetic Programming Theory and Practice VII*, pages 211–228. Springer, 2010.

Michael F Korns. Abstract expression grammar symbolic regression. In *Genetic Programming Theory and Practice VIII*, pages 109–128. Springer, 2011.

Michael F Korns. A baseline symbolic regression algorithm. In *Genetic Programming Theory and Practice X*, pages 117–137. Springer, 2013.

Michael F Korns. Extreme accuracy in symbolic regression. In *Genetic Programming Theory and Practice XI*, pages 1–30. Springer, 2014.

Michael F Korns. Extremely accurate symbolic regression for large feature problems. In *Genetic Programming Theory and Practice XII*, pages 109–131. Springer, 2015.

Michael F Korns and Loryfel Nunez. Profiling symbolic regression-classification. In *Genetic Programming Theory and Practice VI*, pages 1–14. Springer, 2009.

Mark Kotanchek and Nathan Haut. *Back to the Future—Revisiting OrdinalGP and Trustable Models After a Decade*, pages 129–142. Springer Nature Singapore, Singapore, 2022. ISBN 978-981-16-8113-4. doi: 10.1007/978-981-16-8113-4_7. URL https://doi.org/10.1007/978-981-16-8113-4_7.

Mark Kotanchek, Guido Smits, and Ekaterina Vladislavleva. Pursuing the pareto paradigm: Tournaments, algorithm variations, and ordinal optimization. In Rick Riolo, Terence Soule, and Bill Worzel, editors, *Genetic Programming Theory and Practice IV*, pages 167–185. Springer, 2007.

Mark Kotanchek, Guido Smits, and Ekaterina Vladislavleva. Exploiting trustable models via pareto gp for targeted data collection. In Rick Riolo, Terence Soule, and Bill Worzel, editors, *Genetic*

Programming Theory and Practice VI, pages 145–162. Springer, 2009.

John R Koza. *Genetic Programming: On the Programming of Computers By Means Of Natural Selection*. MIT Press, Cambridge, MA, 1992a.

John R. Koza. *Genetic Programming — On the Evolution of Computer Programs by Means of Natural Selection.*, volume 1. MIT Press, Cambridge, MA., 1992b.

Jan Kremer, Kim Steenstrup Pedersen, and Christian Igel. Active learning with support vector machines. *WIREs Data Mining and Knowledge Discovery*, 4(4):313–326, 2014.

William La Cava, Patryk Orzechowski, Bogdan Burlacu, Fabrício Olivetti de França, Marco Virgolin, Ying Jin, Michael Kommenda, and Jason H. Moore. Contemporary symbolic regression methods and their relative performance, 2021. URL https://arxiv.org/abs/2107.14351.

Christian W. Lasarczyk, Peter Dittrich, and Wolfgang Banzhaf. Dynamic subset selection based on a fitness case topology. *Evolutionary Computation*, 12(2):223 – 242, 2004.

David D. Lewis and William A. Gale. A sequential algorithm for training text classifiers. In Bruce W. Croft and C. J. van Rijsbergen, editors, *SIGIR '94*, pages 3–12, London, 1994. Springer London. ISBN 978-1-4471-2099-5.

Changsheng Li, Xiangfeng Wang, Weishan Dong, Junchi Yan, Qingshan Liu, and Hongyuan Zha. Joint active learning with feature selection via cur matrix decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41:1382–1396, 2015. URL https://api.semanticscholar.org/CorpusID:17165137.

Yuanzhang Li, Xinxin Wang, Zhiwei Shi, Ruyun Zhang, Jingfeng Xue, and Zhi Wang. Boosting training for pdf malware classifier via active learning. *International Journal of Intelligent Systems*, 37(4):2803–2821, 2022.

D. V. Lindley. On a Measure of the Information Provided by an Experiment. *The Annals of Mathematical Statistics*, 27(4):986 – 1005, 1956.

Manuel Lopes, Francisco Melo, and Luis Montesano. Active learning for reward estimation in inverse reinforcement learning. In Wray Buntine, Marko Grobelnik, Dunja Mladenić, and John Shawe-Taylor, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 31–46, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-04174-7.

James McDermott, David R White, Sean Luke, Luca Manzoni, Mauro Castelli, Leonardo Vanneschi, Wojciech Jaskowski, Krzysztof Krawiec, Robin Harper, Kenneth De Jong, et al. Genetic programming needs better benchmarks. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, pages 791–798, 2012.

Julian F. Miller. *Cartesian Genetic Programming*, pages 17–34. Springer Berlin Heidelberg, Berlin,

Heidelberg, 2011.

Tom M. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203–226, 1982. ISSN 0004-3702. doi: https://doi.org/10.1016/0004-3702(82)90040-6. URL https://www.sciencedirect.com/science/article/pii/0004370282900406.

Berndt Müller, Joachim Reinhardt, and Michael T Strickland. *Neural networks: an introduction*. Springer Science & Business Media, 1995.

Aditya Nandy, Chenru Duan, Conrad Goffinet, and Heather J. Kulik. New strategies for direct methane-to-methanol conversion from active learning exploration of 16 million catalysts. *Journal of the American Chemical Society Au*, 2(5):1200–1213, 2022.

Feiping Nie, Hua Wang, Heng Huang, and Chris Ding. Early active learning via robust representation and structured sparsity. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, page 1572–1578. AAAI Press, 2013. ISBN 9781577356332.

Nutonian. Eureqa. URL https://www.datarobot.com/nutonian/.

J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986. doi: 10.1007/BF00116251. URL https://doi.org/10.1007/BF00116251.

Carl Rasmussen. The infinite gaussian mixture model. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999. URL https://proceedings.neurips.cc/paper_files/paper/1999/file/97d98119037c5b8a9663cb21fb8ebf47-Paper.pdf.

Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Brij B. Gupta, Xiaojiang Chen, and Xin Wang. A survey of deep active learning. *ACM Comput. Surv.*, 54(9), oct 2021.

Brianna Ricker, Sunayana Mitra, E. Alejandro Castellanos, Connor J. Grady, Galit Pelled, and Assaf A. Gilad. Proposed three-phenylalanine motif involved in magnetoreception signaling of an actinopterygii protein expressed in mammalian cells. *bioRxiv*, 2022. doi: 10.1101/2022.12.08.519643. URL https://www.biorxiv.org/content/early/2022/12/09/2022.12.08.519643.

Ognjen Rudovic, Meiru Zhang, Bjorn Schuller, and Rosalind Picard. Multi-modal active learning from human data: A deep reinforcement learning approach. In *2019 International Conference on Multimodal Interaction*, ICMI '19, page 6–15, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450368605. doi: 10.1145/3340555.3353742. URL https://doi.org/10.1145/3340555.3353742.

SciPy. scipy.optimize.differential_evolution - scipy v1.11.1 manual, Last Access 2023a. URL https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.differential_evolution.html.

SciPy. scipy.optimize.minimize - scipy v1.11.1 manual, Last Access 2023b. URL https://docs.

scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html.

H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, page 287–294, New York, NY, USA, 1992. Association for Computing Machinery. ISBN 089791497X. doi: 10.1145/130385.130417. URL https://doi.org/10.1145/130385.130417.

Dinggang Shen, Guorong Wu, and Heung-Il Suk. Deep learning in medical image analysis. *Annual Review of Biomedical Engineering*, 19(1):221–248, 2017.

Lee Spector. PushGP. URL http://faculty.hampshire.edu/lspector/push.html.

Lee Spector. Assessment of problem modality by differential performance of lexicase selection in genetic programming: A preliminary report. In *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '12, page 401–408, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450311786. doi: 10.1145/2330784.2330846. URL https://doi.org/10.1145/2330784.2330846.

Heung-Il Suk and Dinggang Shen. *Deep Learning in Diagnosis of Brain Disorders*, pages 203–213. Springer Netherlands, Dordrecht, 2015. ISBN 978-94-017-7239-6. doi: 10.1007/978-94-017-7239-6_14. URL https://doi.org/10.1007/978-94-017-7239-6_14.

Yuriy Sverchkov and Mark Craven. A review of active learning approaches to experimental design for uncovering biological networks. *PLOS Computational Biology*, 13:1–26, 6 2017.

Max Tegmark. Welcome to the Feynman Symbolic Regression Database! URL https://space.mit.edu/home/tegmark/aifeynman.html.

Hideaki Uchiyama, Shunsuke Sakurai, Masashi Mishima, Daisaku Arita, Takashi Okayasu, Atsushi Shimada, and Rin ichiro Taniguchi. KOMATSUNA Dataset. URL https://limu.ait.kyushu-u.ac.jp/~agri/komatsuna/.

S.-M. Udrescu and Tegmark M. A physics-inspired method for symbolic regression. *Science Advances*, 6:eaay2631, 2020a.

S.-M. Udrescu and Tegmark M. A physics-inspired method for symbolic regression. *Science Advances*, 6:eaay2631, 2020b.

Nguyen Quang Uy, Nguyen Xuan Hoai, Michael O'Neill, Robert I McKay, and Edgar Galván-López. Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines*, 12(2):91–119, 2011.

Vladimir Vapnik. *Falsifiability and Parsimony: VC Dimension and the Number of Entities (1980–2000)*, pages 425–457. Springer New York, New York, NY, 2006. ISBN 978-0-387-34239-9. doi: 10.1007/0-387-34239-7_12. URL https://doi.org/10.1007/0-387-34239-7_12.

Ekaterina J Vladislavleva, Guido F Smits, and Dick Den Hertog. Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. *IEEE Transactions on Evolutionary Computation*, 13(2):333–349, 2008.

Jana Wäldchen and Patrick Mäder. Plant species identification using computer vision techniques: A systematic literature review. *Archives of Computational Methods in Engineering*, 25(2):507–543, 2018. doi: 10.1007/s11831-016-9206-z. URL https://doi.org/10.1007/s11831-016-9206-z.

David R White, James Mcdermott, Mauro Castelli, Luca Manzoni, Brian W Goldman, Gabriel Kronberger, Wojciech Jaśkowski, Una-May O'Reilly, and Sean Luke. Better gp benchmarks: community survey results and proposals. *Genetic Programming and Evolvable Machines*, 14(1): 3–29, 2013.

Kai Yu, Jinbo Bi, and Volker Tresp. Active learning via transductive experimental design. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, page 1081–1088, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595933832. doi: 10.1145/1143844.1143980. URL https://doi.org/10.1145/1143844.1143980.

APPENDIX A

Table A.1 and Table A.2 show the full results for all 100 of the Feynman symbolic regression problems using the Mathematica implementation. Table A.3 shows the full results on the non-trivial yet reasonably solvable problems using the Python version.

EQ Num	AIFeynman Data Pts	StackGP Pts (num trials)	StackGP Success	Eureqa Success
1	10	3 (100)	Yes	No
2	100	43 (100)	Yes	No
3	1000	42.5 (100)	Yes	No
4	100	25 (100)	Yes	No
5	1000000	- (100)	No	No
6	10	- (100)	No	No
7	100	>102 (100)	No	Yes
8	10	3 (100)	Yes	Yes
9	10	68 (1)	Yes	Yes
10	10	4 (100)	Yes	Yes
11	10	3 (100)	Yes	Yes
12	10	3 (100)	Yes	Yes
13	10	21 (1)	Yes	Yes
14	10	19.5 (100)	Yes	Yes
15	10	3 (100)	Yes	Yes
16	10	3 (100)	Yes	Yes
17	10	-	No	No
18	100	-	No	No
19	10	-	No	No
20	10	-	No	No
21	10	-	No	Yes
22	10	3 (100)	Yes	Yes
23	10	4 (100)	Yes	Yes
24	10	28.5 (100)	Yes	Yes
25	10	3 (100)	Yes	Yes
26	100	3 (100)	Yes	Yes
27	10	12 (100)	Yes	Yes
28	10	3 (100)	Yes	Yes
29	1000	-	No	No
30	100	-	No	Yes
31	100	3 (1)	Yes	Yes
32	10	10.5 (100)	Yes	Yes
33	10	-(100)	No	No
34	10	3 (1)	Yes	Yes
35	10	4 (1)	Yes	No
36	10	-	No	No
37	10	3 (1)	Yes	Yes
38	100	-	No	Yes
39	10	11 (1)	Yes	Yes
40	10	3 (1)	Yes	Yes
41	10	9 (1)	Yes	Yes
42	10	3 (1)	Yes	Yes
43	10	102 (1)	Yes	No
44	10	-	No	No
45	10	3 (1)	Yes	Yes
46	10	3 (1)	Yes	Yes
47	10	28.5 (40)	Yes	Yes
48	10	32 (1)	Yes	Yes
49	10	3 (1)	Yes	Yes
50	100	-	No	No

Table A.1 Number of Data Points Needed to Solve Problems 1-50. For The StackGP solution, the number of points is the median of points used out of the indicated number of trials. For Eureqa, 300 points were used for each equation with a 2 hour time limit.

EQ Num	AIFeynman Data Pts	StackGP Pts (num trials)	StackGP Success	Eureqa Success
51	10	-	No	Yes
52	10	13 (1)	Yes	Yes
53	10	3 (1)	Yes	Yes
54	10	3 (1)	Yes	Yes
55	10	13 (1)	Yes	Yes
56	10000	-	No	No
57	10	92 (1)	Yes	Yes
58	10	3 (1)	Yes	Yes
59	10	3 (1)	Yes	Yes
60	10	8 (100)	Yes	Yes
61	10	30 (100)	Yes	Yes
62	10	76.5 (100)	Yes	Yes
63	10	14 (1)	Yes	Yes
64	100	-	No	No
65	100	-	No	No
66	10	5 (100)	Yes	Yes
67	100	68 (1)	Yes	No
68	10	-	No	No
69	10	3 (1)	Yes	Yes
70	10	3 (1)	Yes	Yes
71	10	22 (1)	Yes	Yes
72	10	-	No	No
73	10	3 (100)	Yes	Yes
74	10	3 (100)	Yes	Yes
75	10	3 (100)	Yes	Yes
76	10	3 (100)	Yes	Yes
77	10	3 (100)	Yes	Yes
78	10	3 (100)	Yes	Yes
79	10	3 (1)	Yes	Yes
80	10	-	No	No
81	10	-	No	Yes
82	10	-	No	Yes
83	10	4 (100)	Yes	Yes
84	10	3 (100)	Yes	Yes
85	10	4 (1)	Yes	Yes
86	10	-	No	No
87	10	-	No	No
88	10	3 (1)	Yes	Yes
89	10	6 (1)	Yes	No
90	1000	-		No
91	100	83 (1)	Yes	Yes
92	10	3 (1)	Yes	Yes
93	10	5 (1)	Yes	Yes
94	10	-	No	No
95	10	10 (1)	Yes	Yes
96	10	3 (100)	Yes	Yes
97	10	3 (100)	Yes	Yes
98	10	7 (1)	Yes	Yes
99	10	9 (1)	Yes	Yes
100	10	3 (100)	Yes	Yes

Table A.2 Number of Data Points Needed to Solve Problems 51-100. For The StackGP solution, the number of points is the median of points used out of the indicated number of trials. For Eureqa, 300 points were used for each equation with a 2 hour time limit.

EQ Num	Mathematica Data Pts	Python Data Pts
2	43	55
3	42.5	>102
4	25	20
7	>102	21
9	68	101
10	4	11
13	21	15
14	19.5	24
23	4	8
24	28.5	29.5
27	12	13
32	10.5	18
35	4	6
39	11	12
41	9	8
43	102	81
47	28.5	12
48	32	17
52	13	10
55	13	12
57	92	27
60	8	7
61	30	19
62	76.5	34.5
63	14	16
66	5	14
67	68	10
71	22	58
83	4	5
85	4	4
89	6	5
91	83	>102
93	5	8
95	10	11.5
98	7	9
99	9	35.5

Table A.3 Number of Data Points Needed to Solve Non-Trivial Problems for Python and Mathematica Implementations.

APPENDIX B

This appendix lists all 100 AI Feynman problems and their solution using correlation and RMSE as fitness functions, for 0% and 10% noise levels at 3 data points only.

Filename	EQ #	Correlation 3 Pts 0%	RMSE 3 Pts 0%	Correlation 3 Pts 10%	RMSE 3 Pts 10%
I.6.20a	1.	**0.032**	0.304	**0.12**	0.35
I.6.20	2.	**0.41**	0.53	**0.41**	0.43
I.6.20b	3.	**0.68**	0.83	**0.57**	0.61
I.8.14	4.	21.71	**21.61**	**11.2**	15.66
I.9.18	5.	**2.02**	2.09	**2.15**	2.24
I.10.7	6.	2.29	**1.63**	2.74	**1.95**
I.11.19	7.	155.4	**137.08**	145.9	**142.88**
I.12.1	8.	0.	0.	3.74	**0.**
I.12.2	9.	291.77	**235.69**	302.15	**244.97**
I.12.4	10.	1.29	**1.17**	1.25	**0.95**
I.12.5	11.	0.29	**0.26**	0.29	**0.26**
I.12.11	12.	0.	0.	4.48	**0.**
I.13.4	13.	**311.99**	382.06	380.01	**378.98**
I.13.12	14.	123.88	**123.**	131.49	**121.92**
I.14.3	15.	0.	0.	16.67	**0.**
I.14.4	16.	**0.**	29.75	**9.03**	29.53
I.15.3x	17.	9.06	**6.44**	9.83	**7.16**
I.15.3t	18.	2.15	**1.81**	2.31	**2.1**
I.15.1	19.	4.34	**3.57**	4.91	**3.54**
I.16.6	20.	**7.26**	8.16	**8.41**	9.61
I.18.4	21.	**5.2**	6.38	**5.73**	6.4
I.18.12	22.	**39.64**	81.53	88.04	**0.**
I.18.14	23.	289.04	**269.43**	312.51	**294.09**
I.24.6	24.	186.77	**103.88**	172.39	**111.05**
I.25.13	25.	0.	0.	0.61	**0.**
I.26.2	26.	1.44	**0.**	4.78	**0.**
I.27.6	27.	4.59	**3.53**	**3.47**	3.82
I.29.4	28.	0.	0.	0.88	**0.**
I.29.16	29.	42.96	**39.75**	44.78	**39.01**
I.30.3	30.	170.3	**47.**	56.17	**45.07**
I.30.5	31.	0.	0.	1.07	**0.**
I.32.5	32.	**10.48**	11.09	11.02	**10.95**
I.32.17	33.	**58.75**	62.13	74.19	**63.58**
I.34.8	34.	118.78	**117.31**	152.25	**112.47**
I.34.1	35.	12.5	**12.39**	12.833	**12.611**
I.34.14	36.	9.69	**6.41**	9.27	**8.09**
I.34.27	37.	0.	0.	4.64	**0.**
I.37.4	38.	**45.37**	53.82	**34.71**	44.55
I.38.12	39.	809.61	**764.83**	797.34	**766.62**
I.39.1	40.	**0.**	33.53	**6.12**	49.26
I.39.11	41.	40.23	**22.69**	32.95	**29.32**
I.39.22	42.	**57.45**	109.84	136.6	**116.42**
I.40.1	43.	3.21×10^{12}	3.12×10^{12}	2.3×10^{15}	9.67×10^{14}
I.41.16	44.	31.63	**29.5**	31.62	**30.15**
I.43.16	45.	**93.43**	123.12	140.63	**113.85**
I.43.31	46.	0.	0.	15.21	**0.**
I.43.43	47.	24.15	**20.21**	23.18	**19.87**
I.44.4	48.	315.51	**278.57**	299.86	**265.23**
I.47.23	49.	**5.43**	6.32	7.34	**4.88**
I.48.2	50.	30.91	**17.23**	136.99	**29.78**

Table B.1 The resulting RMSE values on test data when using correlation or RMSE as the fitness function during training for the first 50 Feynman equations. Three random training data points were used with either 0 or 10% noise.

Filename	EQ #	Correlation 3 Pts 0%	RMSE 3 Pts 0%	Correlation 3 Pts 10%	RMSE 3 Pts 10%
I.50.26	51.	**32.39**	32.81	35.81	**33.07**
II.2.42	52.	96.89	**67.68**	111.34	**93.28**
II.3.24	53.	**0.**	0.22	**0.044**	0.328
II.4.23	54.	**0.**	0.29	**0.339**	0.357
II.6.11	55.	**0.27**	0.28	**0.279**	0.362
II.6.15a	56.	5.59	**4.05**	5.01	**4.11**
II.6.15b	57.	40.16	**39.29**	29.19	**28.66**
II.8.7	58.	**1.06**	1.24	**1.106**	1.116
II.8.31	59.	**0.**	35.63	**9.68**	23.01
II.10.9	60.	2.97	**1.52**	2.74	**1.79**
II.11.3	61.	**1.135**	1.155	1.25	**1.08**
II.11.17	62.	27.32	**24.32**	**20.98**	22.98
II.11.20	63.	118.25	**94.68**	103.66	**95.24**
II.11.27	64.	5.24	**2.61**	4.21	**2.71**
II.11.28	65.	**0.012**	0.2	0.77	**0.52**
II.13.17	66.	**0.3047**	0.3393	0.36	**0.3**
II.13.23	67.	2.12	**1.59**	2.68	**1.93**
II.13.34	68.	3.9	**2.76**	5.75	**3.76**
II.15.4	69.	**67.4**	88.35	95.53	**83.07**
II.15.5	70.	**63.28**	80.92	94.52	**92.21**
II.21.32	71.	0.714	**0.656**	0.67	**0.66**
II.24.17	72.	**1.72**	2.31	**2.405**	2.963
II.27.16	73.	0.	0.	79.1	**0.**
II.27.18	74.	0.	0.	17.51	**0.**
II.34.2a	75.	**0.**	5.86	**4.33**	5.04
II.34.2	76.	**0.**	47.1	**9.4**	45.
II.34.11	77.	65.58	**62.45**	73.52	**63.65**
II.34.29a	78.	**0.**	3.07	**1.98**	2.996
II.34.29b	79.	569.68	**454.52**	534.74	**460.95**
II.35.18	80.	**4.73**	5.1	**4.99**	5.36
II.35.21	81.	65.37	**49.**	58.39	**48.6**
II.36.38	82.	18.52	**16.24**	18.7	**15.76**
II.37.1	83.	136.845	**46.77**	92.81	**45.08**
II.38.3	84.	122.17	**114.56**	148.47	**98.47**
II.38.14	85.	**0.76**	0.99	**0.88**	1.09
III.4.32	86.	11.23	**7.24**	9.05	**8.04**
III.4.33	87.	41.59	**27.14**	42.31	**33.12**
III.7.38	88.	**0.**	50.43	55.54	**54.95**
III.8.54	89.	**3.68**	4.38	**4.1**	4.6
III.9.52	90.	45.78	**38.94**	46.29	**39.17**
III.10.19	91.	69.8	**54.84**	72.05	**56.28**
III.12.43	92.	0.	0.	4.08	**0.**
III.13.18	93.	1152.36	**1107.82**	1206.76	**1089.88**
III.14.14	94.	109.04	**98.87**	110.38	**97.74**
III.15.12	95.	105.83	**105.45**	**107.98**	109.96
III.15.14	96.	8.06	**7.1**	7.84	**7.08**
III.15.27	97.	**0.**	33.66	**28.29**	35.75
III.17.37	98.	88.19	**72.66**	90.	**74.54**
III.19.51	99.	**0.61**	0.72	**0.59**	0.86
III.21.20	100.	**126.007**	131.52	144.83	**132.95**

Table B.2 The resulting RMSE values on test data when using correlation or RMSE as the fitness function during training for the second set of 50 Feynman equations. Three random training data points were used with either 0 or 10% noise.

APPENDIX C

This appendix includes the full comparison of each uncertainty and diversity metric from Chapter 6.

EQ Num	U. Rand Data Pts.	Pt. Dist Data Pts.	Pt. Corr Data Pts.
2	54.5	**44**	-
3	> 1000	> 1000	> 1000
4	30	**19**	29.5
7	88.5	35.5	60
9	120.5	210.5	**102.5**
10	6	6	**5**
13	13	**12**	14
14	30.5	24	**22**
23	8	**7**	8
24	49	**23**	26.5
27	30	13.5	**11.5**
32	17	15.5	**14**
35	19	**13.5**	15
39	10	11	**9**
41	7	8	**6**
43	453	533.5	**136.5**
47	**13**	14.5	16
48	15.5	**13**	**13**
52	9.5	10	**9**
55	10	10	10
57	30.5	**28**	29.5
60	7	7	7
61	18.5	**17.5**	**17.5**
62	34.5	**29.5**	36.5
63	14	14	**12**
66	11	12	**10**
67	**10.5**	11	15
71	51.5	**29**	30.5
83	5	5	5
85	4	4	-
89	5	5	5
93	8	7.5	**7**
95	11	10	**8**
98	8	8	**7**
99	30	25	**20.5**
Perf. Count	-	27/35	29/33

Table C.2 Shown are the median number of points needed to solve each equation. A total of 100 independent trials were performed for each equation. There are 2 equations that have a dash instead of a number and that is because they have only two dimensions, so selecting points with minimal correlation to the rest of the training set is not possible. The approach using uniformly random data points was included in the first column represented as a baseline. The last row indicates the number of cases where each of the point diversity methods matched or performed better than the random approach.

EQ Num	Pt. Dist Data Pts.	Pareto Data Pts.	Pt. Unc. Data Pts.
2	44	36.5	82.5
3	> 1000	501	> 1000
4	19	29	28
7	35.5	48.5	52.5
9	210.5	304	153
10	6	6	6
13	12	12	12
14	24	22	24
23	7	8	7.5
24	23	27	22
27	13.5	19	14
32	15.5	14	18
35	13.5	14	13.5
39	11	10	9
41	8	7	7
43	533.5	314.5	326
47	14.5	12	12
48	13	13	13
52	10	10	9
55	10	9	10
57	28	24	23
60	7	7	7
61	17.5	15	16
62	29.5	21.5	30
63	14	14	13
66	12	10	10
67	11	11	11
71	29	50.5	30
83	5	5	5
85	4	4	4
89	5	5	5
93	7.5	8	8
95	10	13	10
98	8	8	7
99	25	25.5	24
Worst Count	13	11	8
Best Count	16	19	21

Table C.3 Shown are the median number of points needed to solve each equation. A total of 100 independent trials were performed for each equation. Here the trade-off between diversity and uncertainty is explored. The second to last row indicates the number of times each approach was the worst of the three approaches. The last row indicates the number of cases where each approach was the best or tied for the best of the three approaches. Minimum point distance was used for the diversity metric and differential entropy was used as the uncertainty metric.

EQ Num	p-value	Significant
2	$9.18077*10^{-4}$	Yes
3	$1.4012505382465074*10^{-11}$	Yes
4	0.612865	No
7	$9.606121209101481*10^{-6}$	Yes
9	$1.78579*10^{-2}$	Yes
10	$3.40099*10^{-5}$	Yes
13	$2.38254*10^{-5}$	Yes
14	$7.46075*10^{-4}$	Yes
23	$3.3175302779045554*10^{-6}$	Yes
24	$8.64911*10^{-5}$	Yes
27	0.0354696	Yes
32	$1.30021*10^{-3}$	Yes
35	$2.90356*10^{-3}$	Yes
39	$7.61664*10^{-3}$	Yes
41	$5.872949580996268*10^{-7}$	Yes
43	$3.38224*10^{-3}$	Yes
47	$1.91012*10^{-4}$	Yes
48	$1.78267*10^{-5}$	Yes
52	0.433203	No
55	$1.98428*10^{-4}$	Yes
57	$2.51679*10^{-5}$	Yes
60	$6.84394*10^{-3}$	Yes
61	$1.39827*10^{-3}$	Yes
62	$2.6897830944058467*10^{-20}$	Yes
63	0.0399119	Yes
66	$1.55871*10^{-3}$	Yes
67	0.245499	No
71	0.392183	No
83	$9.35933923552931*10^{-10}$	Yes
85	$1.059722812150408*10^{-12}$	Yes
89	0.140942	No
93	$1.54882*10^{-5}$	Yes
95	0.0345741	Yes
98	$2.98341*10^{-3}$	Yes
99	$5.63906*10^{-3}$	Yes
Significance Count	**30/35**	

Table C.4 Statistical significance of Pareto AL approach vs. uniform random sampling. We are using a threshold of 0.05 to test for significance. The Mann-Whitney test was used to test for significance.