Performance Testing of Automated Modeling for Industrial Applications

Dylan Sherry
Nutonian Inc.
125 Summer Street, Suite 1000
Boston, Massachusetts 02210
dylan.sherry@nutonian.com

Michael Schmidt
Nutonian Inc.
125 Summer Street, Suite 1000
Boston, Massachusetts 02210
michael.schmidt@nutonian.com

ABSTRACT

We present a case study of the performance testing of a commercially engineered genetic programming algorithm applied to the automated modeling of industrial machine learning problems. This paper summarizes some of what has been learned over the past five years of working with a large number of industrial machine learning challenges in a commercial or enterprise setting. Automation and parallelism via cloud computing is used to reduce test time. Two frameworks for conducting performance tests are discussed, highlighting the advantages of collecting statistics throughout the search. A performance test suite of industrial machine learning problems is described, and examples of performance test results are shown. Finally, a summary of challenges and open questions is provided.

KEYWORDS

Genetic programming, machine learning, performance test, benchmark, case study

ACM Reference format:
Dylan Sherry and Michael Schmidt. 2017. Performance Testing of Automated Modeling for Industrial Applications. In *Proceedings of GECCO '17 Companion, Berlin, Germany, July 15-19, 2017*, 8 pages.
DOI: http://dx.doi.org/10.1145/3067695.3082534

1 INTRODUCTION

1.1 Assessing Performance is Important

A recent critique of machine learning remarked that "models are opinions embedded in mathematics … A key component of any model … is its definition of success" [12, p. 21]. The definition of model quality may vary depending on the problem domain, and choosing a definition can have important philosophical, social and physical implications [12]. For instance, if a model is to be used to classify whether or not a young child needs emergency surgery, it becomes incredibly important that models do not miss any positive cases and erroneously advocate against surgery for a child in dire need of it [1, 4]. If a model is intended to predict the load of a regional electric power grid, increasing the predictive accuracy by a small percentage could reduce both wasted energy and unnecessary carbon emissions, and also yield huge monetary savings [6].

At Nutonian Inc., tracking the performance of our machine learning software is essential to our users' success and to our business goals. Nutonian's primary product is Eureqa, a software application which streamlines data analysis by automating the process of model building and interpretation, providing a data science workflow which has a high impact and a low barrier of entry for our customers.

Eureqa has undergone substantial modification and enhancement since its inception. An early version of the algorithm was run for 30 hours to find the physical laws governing a double pendulum system [15]. Using the current version of Eureqa, that result can be replicated in seconds. Since one of the technologies used internally by Eureqa is genetic programming (GP) [9, 15, 16], the journey to achieve that performance boost has necessitated a focus on establishing effective performance tests of GP on machine learning problems.

1.2 Assessing Performance is Difficult

There are several attributes of the performance of GP which are important to the experience of Eureqa's users:

- Low test error: How accurate are the models? How well do they generalize to other data (avoiding overfitting and underfitting)?
- Complexity: How simple are the models?
- Interpretability: How easy is it to interpret the models and understand their structure?
- Speed: How long does it take to obtain good results?
- Consistency and variability: What kinds of models appear at different points in the search process? Will a similar trend occur if the search is repeated?
- Cost: How expensive is it to Nutonian to run a search?

The first three elements encapsulate the quality of the models themselves, and the last three concern the quality of the search process to Nutonian and to Eureqa's users. Each of these attributes presents its own set of challenges.

Search algorithms like GP are susceptible to learning overfit or underfit models which don't generalize well outside of training data [2, 14]. Fair performance tests of GP must address this issue.

A search with a high speed to achieve an expected result may do so at the expense of other desirable attributes like model diversity or consistency. The measurement of computational speed may depend on the computational substrate. Servers provisioned via

GECCO '17 Companion, Berlin, Germany

DOI: http://dx.doi.org/10.1145/3067695.3082534

cloud computing frameworks may have a variable computation speed [17, 18].

Assessing performance using industrial machine learning problems can "require a great deal of domain knowledge, and so seem unsuitable as GP benchmarks" [19, p. 24]. We question the assertion that the cost of learning enough of the domain expertise required to construct an effective performance test outweighs the benefits. We have learned enough domain expertise through engagement with users of Eureqa on select problems to gauge when results represent a success, and therefore we feel confident using them as part of performance test suites.

The data and problems faced by machine learning in an industrial setting differ from many previously proposed benchmarks in the GP community. Industrial machine learning problems typically have some degree of noise and no known perfect analytic solution. For some problems a minuscule improvement in accuracy could represent a huge impact in the problem domain. All of these factors may place a different pressure on a GP learning algorithm than synthetic data, where the signal is known a priori and the noise is artificial. It has been observed benchmarks which take this into account by focusing on measuring model quality on harder problems would likely prove valuable to the field [11]. We describe in this paper the value these kinds of performance tests have had for us.

1.3 Summary

This paper details some of the work done at Nutonian to test the performance of GP. We discuss general methods for conducting performance tests of a GP search algorithm on machine learning problems, including the merits of automation and parallelism in testing. We outline two frameworks we have explored for conducting performance testing and analyzing the results, and discuss their relative strengths and weaknesses. Finally we describe some of the challenges we have encountered and provide thoughts on related open questions and directions for future work.

x

2 BACKGROUND

2.1 Related Work

Benchmarking has been raised as an important issue for the GP research community, and one for which there is at present no widely accepted standard [11, 13].

Prior work includes a discussion of scope and values of performance testing. Zitzler et al. state the performance of a multi-objective optimization system consists of two primary components: the quality of the models found during the search process, and the rate at which the results were obtained [20].

Recent work provides a comprehensive survey of previous efforts to define commonly accepted GP benchmarks [11, 19]. Discussion has found the previously accepted synthetic benchmarks [9, 10] as being "too easy" and consisting of "simple toy problems" [19, p. 19]. Assessing performance using industrial machine learning problems can "require a great deal of domain knowledge, and so seem unsuitable as GP benchmarks" [19, p. 24].

The UCI machine learning data repository and UCI KDD archive are mentioned as two potential sources for non-synthetic data to be used in benchmarking [19, p. 13].

2.2 Eureqa

Eureqa is a software application which can be used to automate modeling in machine learning problems. One of the technologies used internally by Eureqa is a multi-objective GP implementation [3, 9, 15, 16] which optimizes a tradeoff between model error and model complexity.

Eureqa has an extensive active user base with applications ranging across numerous industries, including use in a wide variety of academic settings. The system has been used to improve the accuracy of scanning electron microscopes [5], help airplane pilots stay focused and aware during crises [7] and avoid unnecessary emergency surgery in young children with appendicitis [1, 4].

2.2.1 Data Splitting. Eureqa partitions the users' data into two regions. The training data is provided to the GP learning algorithm. The validation data is withheld for model selection as a protection against overfitting. To protect against overfitting on the validation data, test data is typically also withheld to check the performance of the models selected using the validation data.

2.2.2 Searches and Models. Searches and models are two of the fundamental entities in Eureqa. A search consists of a problem type, a target variable to model, one or more input variables, and a set of optional advanced parameters with preselected defaults, some of which correspond to traditional GP parameters. One notable advanced parameter is the error metric used by GP; another is the amount of data used for training v.s. that withheld for validation.

As a search runs for a specified time duration, a Pareto front [3] of candidate models is displayed. When the search completes, the final Pareto front remains visible. At all times, one of the models on the Pareto front is selected by Eureqa as the "best" solution, meaning it represents a good tradeoff between model complexity and minimal error.

The displayed Pareto front contains the models which were selected from the GP learning algorithm and which represent a Pareto optimal tradeoff between model complexity and error on the validation data.

2.2.3 Problem Types. The most common use cases of Eureqa are for solving regression and binary classification problems. The data can optionally be ordered by a time variable. The four default templates for searches in Eureqa are:

- Regression
- Binary classification
- Timeseries regression
- Timeseries classification

The data can also optionally be separated into multiple distinct series. One example of a multiseries machine learning problem would be an experiment where each series encapsulates a single trial; another example is a dataset containing the sales information and other attributes of a retail business, where each series represents a distinct retail outlet.

3 PERFORMANCE TESTING

3.1 Goals and Framework

The goal of the performance tests is to provide quantifiable metrics for measuring the quality of the searches and models produced

by Eureqa's algorithm. This serves as a proxy to a key part of the experience of Eureqa's users. We can then use those metrics to monitor for accidental or unanticipated changes to Eureqa as part of the software development lifecycle, thus providing strong evidence the quality of Eureqa only changes with the knowledge of the development team. We can also use those metrics to determine whether a proposed change to the algorithm is beneficial to model quality and therefore to Eureqa's users.

The general framework of the performance tests is to define a machine learning problem, and measure the effectiveness of the algorithm at finding accurate and meaningful models which satisty the problem. The problem definition consists of the following main components:

- A dataset with one or more input variables and one target variable representing the information Eureqa's users want to understand and predict.
- One of two types of machine learning problems: classification or regression.
- Whether or not the data is ordered by time ("timeseries"), and if so, the variable in the dataset which represents time.
- An error metric to use during training, validation and testing of models.

The performance test suites are a collection of different permutations of those parameters, designed to cover the various ways in which users utilize Eureqa. Each test in the suite will use Eureqa's algorithm to search for solutions to the problem, doing so over multiple trials to normalize for the inherent stochasticity of the search process. The test suite may collect statistics on the models and other properties of the algorithm over the course of the search. After all test trials are complete, the results are displayed and analyzed, and team members can work with that information to make decisions about how to modify Eureqa's algorithm.

3.2 Sequential vs Parallel Testing

The first attempts at performance testing took a straightforward approach by running test trials sequentially on a single cluster. The performance test suites we rely on today run all test trials in parallel. This is achieved through the use of a cloud computing framework like Amazon Web Services Elastic Compute Cloud (AWS EC2), where virtualized servers can be created and kept running for an hourly fee. The parallelized performance tests first generate a large number of clusters, then assign each cluster a test trial to evaluate independently from the others.

3.3 Automation of Performance Tests

3.3.1 Automated Testing. The performance test suites can be automatically queued and run via an automation program called Jenkins. [8] All performance test results are archived and can be accessed via Jenkins by any member of the engineering teams.

To detect any unanticipated performance changes introduced during development, Jenkins was configured to run the performance tests nightly, and to notify the engineering team if a regression was detected. This provided another level of quality assurance for Eureqa.

4 FIXED ACCURACY TESTS

4.1 Motivation

To achieve our performance testing goals, the first implementation of the performance suite measured the total time to reach an expected result on synthetic datasets. The "fixed accuracy" performance tests were inspired by prior work on benchmarking GP. We built a suite of performance tests which were designed to provide comprehensive coverage of the most common use cases of Eureqa.

4.2 Test Procedure

Several datasets were generated by hand and used to construct regression and classification problems. The datasets were generated using various mathematical functions which represented the a priori expected solution to the problem. Searches were then run using those datasets until the a priori solution was found. The total elapsed wall clock time, model evaluations and generations were recorded and the mean value was used as a performance indicator.

A heuristic was chosen to automatically determine when the best solution had converged on the expected a priori model. To achieve this, an upper threshold on the test error was selected for each test. The thresholds were obtained by runnning the search several times by hand and observing what the average test error was once the best solution had converged on the expected form.

Two values were used in the thresholding to determine the stopping point: the error metric used to run the search, and the correlation coefficient between the model output and expected model output. This was done in case the model was nearly identical to the a priori expected mode, but it wasn't linear fitted correctly and produced a poor R^2 or other error metric.

Each performance test iteration was run until the best model selected from the validation Pareto front achieved a test error less than or equal to the specified threshold, or until the trial duration exceeded a max allowed value.

Once the best model of the search reached the test error thresholds and the search was stopped, the elapsed wall clock time, generations and evaluations were recorded.

4.3 Test Suite Description

The fixed-accuracy test suite included the following tests:

- Korns-12, Pagie-1 and Vladislavleva-4 from White et al. [19]
- Three classification tests: one with balanced classes, one with a 19:1 ratio from majority to minority class, and one with a 19:1 ratio and also a 3:1 within-class imbalance in the input space.
- Regression of a simple timeseries signal with moderate Gaussian noise added to the target variable.
- An overspecified regression test, where 100K rows were generated for a simple harmonic function.
- A distractor test, where 1000 random variables were included as inputs, along with a single input which had a nonlinear dependence on the target variable.
- A variety of easy linear and nonlinear functions, with and without a limited operator set.

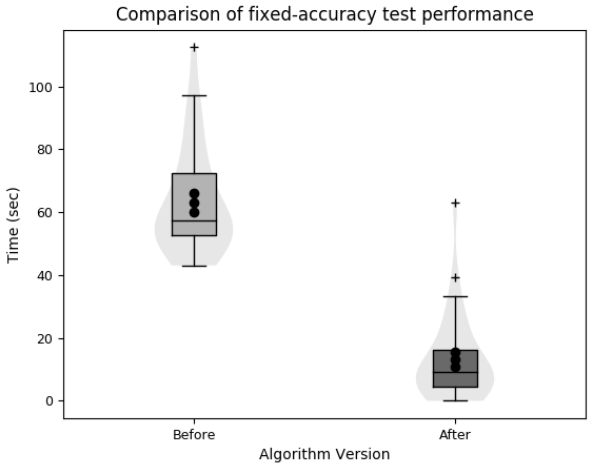

Figure 1: An example of fixed-accuracy performance test results for a particular test, shown before and after a proposed algorithmic change. The middle points represent the mean result, and the upper and lower points show the error bars (standard error). The box plots show the quartiles and the violin plots in the background show the density of results.

- Regression of a periodic function with moderate Gaussian noise added to the target variable.
- Same as previous, but with the operator set limited to basic arithmetic to force a Taylor-series expansion.

All the test data was synthetic. The tests ranged in size from 100 rows to 100K rows, and from 1 input to 1001. Some, like Korns-12 and the distractor problem, required hundreds of iterations to achieve reasonable error bars. Others like the basic arithmetic tests required only about 30 iterations. The tests used the R^2, correlation and area under the ROC curve (AUC) error metrics. The max allowed trial duration was 10min.

4.4 Process for Analyzing Fixed Accuracy Test Results

After performance test results were collected, the statistics for each test were converted to plots showing the distribution of search durations across trials. Figure 1 shows an example of the plots generated by the fixed-accuracy performance tests.

In the particular case depicted, the "after" is shown to represent a positive change relative to the previous results because it greatly reduces the time to reach the expected solution for the performance test.

5 FIXED TIME TESTS

5.1 Motivation

The fixed accuracy tests had several deficiencies. The measurements were all made at the end of each trial, which meant the tests provided no information about the intermediate stages of a search. They had a high variance which required sometimes hundreds of trials to reduce error bars. They required fragile calibration to set the error thresholds for the stopping criterion. There was no guarantee the stopping criterion was achieving its intended purpose of ending the search when the correct solution had been found.

Eureqa's users' data comes from a wide range of sources spanning several industries; the synthetic datasets were not an appropriate reflection of that diversity. In particular, the synthetic datasets were mostly noiseless, and where noise was added it was difficult to know if it reflected the kinds of noise and chaos present in users' data. Many machine learning problems may have no single correct analytic solution which can be drawn from their data.

The fixed time performance test suites were created to address these issues.

5.2 Test Procedure

All of the tests used real world data drawn from customer engagements. Data was selected to cover key problems and usecases. The signals and noise in the data reflected real, potent machine learning challenges which were meaningful to Eureqa's user base.

Each trial was run for a fixed time duration. Measurements were taken periodically throughout the search, approximately every 15 seconds.

The problems had no known correct solution, so the tests defined machine learning benchmarks rather than synthetically constrained "toy problems" [11] like the fixed-accuracy test suites. These tests were not searching for an exact form of solution. As such, no thresholds or heuristics were required as part of the test measurements.

Each measurement recorded a variety of qualities, all of which could be helpful for understanding what impact an algorithmic change has on search performance:

- Elapsed wall clock time.
- Elapsed model evaluations.
- Elapsed mean generations of the GP training populations.
- Model evaluations and generations per second.
- Number of models on the validation Pareto front.
- Mean, standard deviation, median, min and max of the following, computed on the validation Pareto front models:
  - Complexity
  - Training error
  - Validation error
  - Test error
- Complexity of the validation Pareto front best model.
- Training, validation and test error of the validation Pareto front best model.

Figure 2 shows an example of the difference in collection methods between the fixed accuracy and fixed time performance tests. The fixed accuracy tests only record a data point when the best test error crosses below the dotted line, satisfying the stopping criterion. The fixed time tests have no such heuristic and will record data periodically until the search has run for 10 minutes.

5.3 Test Suite Description

The initial fixed-time performance test suite was composed of seven problems, as described in Table 1. The suite was designed to be representative of Eureqa's users' most frequent use cases. Each test

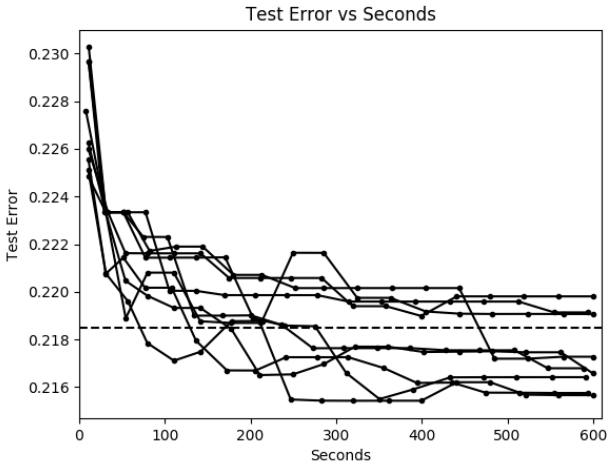

Figure 2: An example of when the fixed-accuracy and fixed-time performance test results are gathered. Each solid black curve represents the test error v.s. time for one trial of a test. The points show the moments in wall-clock time when data was recorded from each trial for fixed time performance tests. The dotted line represents the threshold for a hypothetical fixed accuracy performance test, illustrating that data is recorded only when each trial curve crosses the dotted line, or when 10min has elapsed.

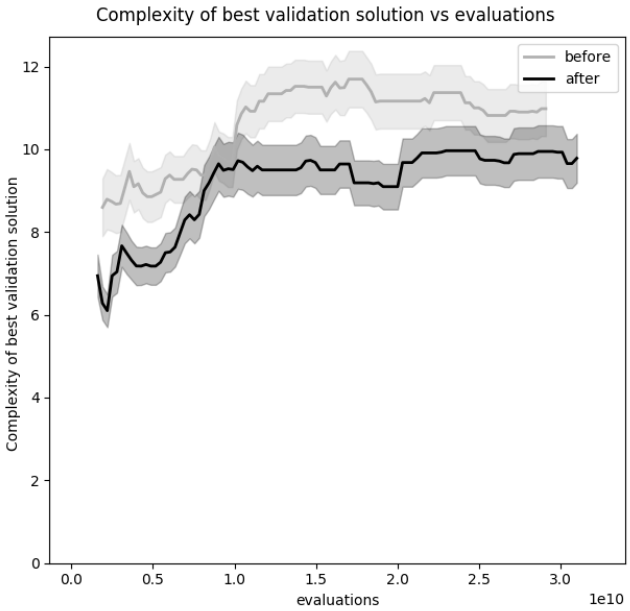

Figure 3: A plot of the complexity of the best model from the validation front v.s. elapsed evaluations. The darker and lighter curves are the results from before and after an algorithm change was applied. The central curves show the mean and the envelope around the curves shows the standard error.

was obtained through a specific customer engagement and defines a problem of importance to the corresponding customer.

5.4 Process for Analyzing Fixed Time Test Results

Once the tests have completed, several statistics and plots can be produced to interpret and understand the results.

The wall clock time, model evaluations and generations can all be used as the x axis for plotting the other results which target different aspects of model quality.

When averaged or plotted against time, the model evaluations per second and generations per second provide a good litmus test for understanding how an algorithmic change effects GP and for confirming the results were collected without obvious errors. For instance, observing the model evaluations per second have dropped for a proposed algorithmic change indicates that the algorithmic change has slowed model evaluation. If that behavior is not consistent with the expected behavior, further review and understanding are required.

The number of models on the validation front and the complexity statistics measure that the algorithm is populating the Pareto front with a diverse spectrum of models representing a tradeoff between model complexity and low error. An example of this is shown in Figure 3 for complexity vs evaluations. In this case, the proposed algorithmic change is shown to reduce the complexity of the best solution from the validation front.

The average training, validation and test error of the validation solutions can be plotted against time to get a general sense of the performance of the Pareto front. However we find it more convincing to examine the error of the best validation solution, since Eureqa recommends the best solution of the search to users as a good model to examine first. Figure 4 shows these numbers for one of the performance tests. In this example the proposed algorithmic change is shown to achieve better test error and therefore outperform the old version.

The results across all tests can be combined to produce an aggregate performance plot which may concisely summarize a performance change across most or all of the tests.

The procedure for combining results is as follows. First a statistic is selected to plot on the y axis. A global min and a max normalization value $normGlobal_{min}$ and $normGlobal_{max}$ are found for that statistic by computing the min and max value of the statistic, across all trials of all tests. That is repeated for each test to find min and max normalization values $norm_{min}[i]$ and $norm_{max}[i]$ for each ith test. Next the data for the statistic in question is accessed from each test, and each set of values $y[i]$ from the ith trial is scaled by the formula:

$$normRangeGlobal = normGlobal_{max} - normGlobal_{min} \quad (1)$$

$$normRange[i] = norm_{max}[i] - norm_{min}[i] \quad (2)$$

$$yScaled[i] = \frac{y[i] - norm_{min}[i]}{normRange[i]} * (normRangeGlobal) + normGlobal_{min} \quad (3)$$

Table 1: The fixed-time performance test suite. 40 trials of each test were run for 10min each. Classification tests used the area under the ROC curve error metric; other tests used mean squared error. The training and validation splits were selected in order for timeseries searches and randomly for others.

Domain	Prob. Type	Rows, Inputs	% Train/Val/Test	Description
Medical	Classification	768, 8	34.5 / 34.5 / 31%	Determine if patients have a disease. 35%/65% class imbalance
Marketing	Classification	10K, 591	33.5 / 33.5 / 33%	Determine if household is a potential customer. Evenly sampled classes.
Manufacturing	Regression	84, 7	33.5 / 33.5 / 33%	Learn to calibrate a complex machine in 8 dimensions
Regional Sales	Timeseries regression	204, 474	52.5 / 22.5 / 25%	Multiseries timeseries regression with 7 series of 34 rows. Each series corresponds to an outlet, and every row represents one month of sales at a regional outlet
Online Marketing	Timeseries regression	154, 153	52.5 / 22.5 / 25%	Learn the aggregate daily online engagement of a major media outlet, over five months, as a function of key events, user behavior, demographics and other information.
Sales Forecasting	Timeseries regression	110, 170	59.5 / 25.5 / 15%	Learn the weekly total sales of a major retailer, over two years, as a function of key events, marketing and promotional events, and other information.
Stock Prediction	Timeseries classification	1194, 10	52.5 / 22.5 / 25%	Determine if a stock had a positive return next month as a function of three years of market indicators and other stocks.

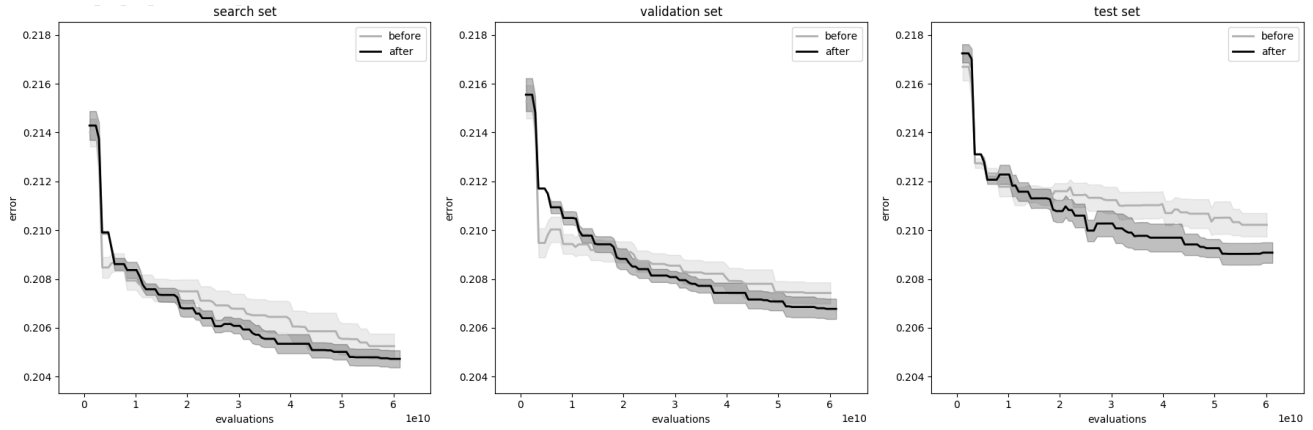

Figure 4: A plot of the training, validation and test error of the best validation solution v.s. elapsed evaluations, across all trials of a particular performance test. The darker and lighter curves are the results from before and after an algorithm change was applied. The central curves show the mean and the envelope around the curves shows the standard error.

The result is then plotted in the same manner as the single performance test plots exemplified in Figure 4.

6 DISCUSSION

6.1 Sequential v.s. Parallel Testing

The biggest advantage of parallelism in performance testing via cloud computing was a great reduction in the computation time, on the order of 10x. This has accelerated the ability to prototype and test changes to the algorithm. The monetary cost per run is nominal, on the order of several US dollars. As an added benefit, the cluster configuration is identical to that used by our production systems, which means the performance tests more accurately reflect Eureqa's users' experiences.

One disadvantage of parallelized tests in this manner is that AWS does not guarantee the effective computational power of each virtual server will remain constant. By running multiple trials we hope to average out any subtle constant difference in computational power across servers; yet it is still possible there could be a non-stationary change in the effective computational power of a cluster during a test.

6.2 Fixed Accuracy v.s. Fixed Time Performance Tests

6.2.1 Fixed Accuracy. The fixed accuracy performance tests were a direct measure of the aspect of user experience we place the highest value on, which is the time to get to a verifiably correct solution. They provided us with basic coverage for catching any unanticipated changes in the algorithm. There is a historical precedent in the field for this method of testing [19, p. 17-18].

There were several drawbacks to the fixed accuracy tests. Each test required the formulation of a synthetic problem. As prior work has shown [11], it is not easy to construct a synthetic dataset which has the complexity, noise and asymmetry of real data.

Finding a reliable automated method to determine when each search had arrived at the expected solution proved difficult. Each of the fixed-accuracy tests required the selection of error thresholds for the stopping criterion heuristic. The heuristic was imperfect. If the error thresholds were too high, we observed models which could satisfy the stopping criterion but not represent a correct solution; if the thresholds were too low, we observed more searches would fall short of reaching the threshold and only stop at the max runtime threshold.

Measuring information only at the end of the search meant the tests didn't provide information about the health of the search while it was running. This could lead us to inadvertently favor changes which sacrifice initial progress. Focusing only on the best solution presents a similar problem, where a prospective algorithmic change could improve the time to reach the expected solution while reducing the diversity and accuracy of other models on the Pareto front. A good performance test would provide some means to investigate these kinds of shifts in algorithm behavior.

Due to the inherent stochasticity of the search process, the fixed accuracy performance tests had high variance, and for some tests hundreds of trials were required to reduce the standard error. There was no good way to handle runs which never satisfied the thresholds and timed out after 10 minutes. An increase in the number of timed out runs may indicate a performance issue, but that is hard to diagnose further without more detailed information.

Since each trial may run up to a max duration, the total time required to run the suite is not deterministic, which makes it hard to plan the quantity of computational resources to allocate and for how long they should be allocated.

6.2.2 Fixed Time. The fixed time performance tests are a significant improvement over the fixed accuracy tests. Performance is measured continuously throughout the progress of the searches. This grants visibility into trends like the training, validation and test error of the best model from the validation front over time, which is helpful for ensuring proper protection against overfitting.

The tests don't need a messy heuristic for gauging whether or not a particular form of solution has been found yet. Any dataset can be used to configure a new fixed performance test without any calibration. It's easy to interpret whether the runs have converged or if the test against a particular dataset needs a longer search time. The tests can be run for a short duration and still produce an approximate result containing useful information for analysis.

In practice the fixed time tests require many fewer iterations to obtain small separable error bars than with the fixed accuracy tests. The observed difference is on the order of tens of iterations for the fixed time tests v.s. hundreds for the fixed accuracy tests.

The fixed time performance tests have no variability in test duration, which allows reservation of massively parallel computational resources for a specific period of time to avoid waste.

6.3 Future Improvements

There are several aspects of the performance testing discussed herein which yield open questions and which would benefit from future exploration and enhancement.

How sensitive is the search algorithm to changes in external (domain-specific) and internal (GP-specific) parameters? Is there an efficient way to produce performance tests which provide coverage of the space of algorithm parameters? A performance test suite could be constructed where parameters are randomly perturbed and the resulting performance examined to catch regressions.

Could variability in the amount of CPU cycles per unit time of clusters hosted by a cloud computing service produce unintended variations in performance measurements? Cloud computing frameworks like Amazon's EC2 do not guarantee the amount of CPU cycles per unit time remains constant. They could theoretically be subject to stationary and non-stationary changes in cycles per unit time which could impact results. Future work could investigate how impactful these fluctuations are and explore ways to work around them. Periodic benchmarking of raw CPU performance could be used to normalize results for these fluctuations.

How can we measure the consistency of what models appear in the search? Are searches finding the same models every time? The consistency of search results is an important quality which are not fully addressed by the current test framework. Covariates and similar model forms can complicate this issue.

How can we reduce the error bars when comparing performance test results? Not only can searches produce differing results of similar quality, but they can also arrive at those results at different rates relative to wall clock time, elapsed evaluations or elapsed generations.

Is there a robust method for measuring the area under the Pareto front (AUP)? Future work could explore algorithms for computing the AUP which are parameter-free and support comparison of the AUP across different trials and across test results from different versions of the GP algorithm.

7 CONCLUSION

This paper has contributed a case study of the construction of performance tests for a GP algorithm with industrial applications. We outlined methods for parallelizing and automating performance testing, and described and discussed two frameworks for testing and analysis. Finally we summarized some challenges encountered and examined implications for future work.

ACKNOWLEDGEMENTS

We would like to thank Hongmin Fan for his efforts in building and maintaining the performance tests, and Andrew Lamb and the rest of the engineering and other teams at Nutonian for their tireless help.

REFERENCES

[1] Einat Blumfield, Gopi Nayak, Ramya Srinivasan, Matthew Tadashi Muranaka, Netta M Blitman, Anthony Blumfield, and Terry L Levin. 2013. JOURNAL CLUB: Ultrasound for Differentiation Between Perforated and Nonperforated Appendicitis in Pediatric Patients. *American Journal of Roentgenology* 200, 5 (2013), 957–962.

[2] Gavin C Cawley and Nicola LC Talbot. 2010. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research* 11, Jul (2010), 2079–2107.

[3] Jeffrey Horn, Nicholas Nafpliotis, and David E Goldberg. 1994. A niched Pareto genetic algorithm for multiobjective optimization. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*. Ieee, 82–87.

[4] Nutonian Inc. 2017. Children's Appendicitis: Eureqa Helps Researchers to Uncover the True Strength of Ultrasound as a Tool for Detecting Appendix Perforation in Children. http://nutonian.com/customers/radnostics/. (2017).

[5] Nutonian Inc. 2017. Instrument Image Distortion: Pierce the Veil of Distortion Over Mission Critical Images. http://nutonian.com/customers/air-force/. (2017).

[6] Nutonian Inc. 2017. Kansas City Power and Light Uses Eureqa to Precisely Match Electric Power Supply to Demand. http://nutonian.com/customers/kansas-city-power-light/. (2017).

[7] Nutonian Inc. 2017. NASA Maximizes Flight Safety With Eureqa. http://nutonian.com/customers/nasa/. (2017).

[8] Kohsuke Kawaguchi. 2014. Meet Jenkins. https://jenkins.io/. (2014).

[9] John R Koza. 1992. *Genetic programming: on the programming of computers by means of natural selection*. Vol. 1. MIT press.

[10] John R Koza. 1994. Genetic programming II: Automatic discovery of reusable subprograms. *Cambridge, MA, USA* (1994).

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

[12] Cathy O'Neil. 2016. *Weapons of math destruction: How big data increases inequality and threatens democracy*. Crown Publishing Group (NY).

[13] Michael O'Neill, Leonardo Vanneschi, Steven Gustafson, and Wolfgang Banzhaf. 2010. Open issues in genetic programming. *Genetic Programming and Evolvable Machines* 11, 3-4 (2010), 339–363.

[14] Cullen Schaffer. 1993. Overfitting avoidance as bias. *Machine Learning* 10, 2 (1993), 153–178. DOI: http://dx.doi.org/10.1007/BF00993504

[15] Michael Schmidt and Hod Lipson. 2009. Distilling free-form natural laws from experimental data. *science* 324, 5923 (2009), 81–85.

[16] Michael Schmidt and Hod Lipson. 2013. Eureqa (version 0.98 beta)[software]. *Nutonian, Somerville, Mass, USA* (2013).

[17] Kalyan Veeramachaneni, Ignacio Arnaldo, Owen Derby, and Una-May O'Reilly. 2015. FlexGP. *Journal of Grid Computing* 13, 3 (2015), 391–407.

[18] Kalyan Veeramachaneni, Owen Derby, Dylan Sherry, and Una-May O'Reilly. 2013. Learning regression ensembles with genetic programming at scale. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. ACM, 1117–1124.

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

[20] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M Fonseca, and Viviane Grunert Da Fonseca. 2003. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on evolutionary computation* 7, 2 (2003), 117–132.