Learning Regression Ensembles with Genetic Programming at Scale

Kalyan Veeramachaneni
Massachusetts Institute of Technology
Cambridge, MA
kalyan@csail.mit.edu

Owen Derby
Massachusetts Institute of Technology
Cambridge, MA
ocderby@csail.mit.edu

Dylan Sherry
Massachusetts Institute of Technology
Cambridge, MA
dsherry@csail.mit.edu

Una-May O'Reilly
Massachusetts Institute of Technology
Cambridge, MA
unamay@csail.mit.edu

ABSTRACT

In this paper we examine the challenge of producing ensembles of regression models for large datasets. We generate numerous regression models by concurrently executing multiple independent instances of a genetic programming learner. Each instance may be configured with different parameters and a different subset of the training data. Several strategies for fusing predictions from multiple regression models are compared. To overcome the small memory size of each instance, we challenge our framework to learn from small subsets of training data and yet produce a prediction of competitive quality after fusion. This decreases the running time of learning which produces models of good quality in a timely fashion. Finally, we examine the quality of fused predictions over the progress of the computation.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

Keywords

Large scale data mining; Machine learning; Ensemble methods

1. INTRODUCTION

Growth in the sizes of both computing resources and datasets have prompted us to scale machine learning using regression supported by genetic programming (GP). On-demand computational resources are now accessible in quantities several orders of magnitude greater than have previously been available. Similarly, the increased availability and cost effectiveness of data storage has allowed the collection of many large datasets for learning algorithms. GP symbolic regression continues to mature as a technique, with the emergence of products such as DataModeler [13] and Eureqa [27].

However, accompanying the emergence of on-demand computation are the establishment of large training datasets. This represents a challenge in the context of GP for two reasons. First, the size of the training dataset may now exceeding the capacity of main memory. And second, the computational expense of the GP learner scales with the quantity of training data, as each member's fitness calculation depends on that member's error on the data.

When exploited by a thoughtfully designed parallelization framework together with efficient scalable decomposition of the computation required for learning, large computational resources can provide the strength to handle large sizes of data. One such method decomposes the data into multiple subsets and learns a large quantity of models via independent learners. This allows the computation to execute on many instances in parallel and is known as a data-parallel approach [17]. Each model is used to make a prediction and a meta-model is developed to fuse these predictions.

In this paper we present a framework which generates ensembles of regression models for large datasets by utilizing large quantities of computational resources teamed with a parallelization framework. We focus on regression through tree-based GP, which generates non-linear symbolic expressions. Our contributions and the challenges we address are:

- **Ensemble Fusion Methods**: We describe and evaluate a suite of different fusion methods for developing an ensemble which fuses multiple continuous-valued predictions. These range from simple averaging to building a meta model. Fusion approaches for both parametric and non-parametric regression approaches are rare in contrast with a plethora of well-explored approaches for classification.
- **Collaborative Solution Building**: We demonstrate how to address a large data problem through factoring. We do so by creating subsets of explanatory variables or by dividing the dataset into smaller datasets, thereby reducing the computational burden on each learner. To improve speed and address memory limitation, we reduce the data size at each instance by a

GECCO'13, July 6–10, 2013, Amsterdam, The Netherlands.

factor of 100 or even 1000. We explore whether factoring generates weak learners which can be fused into an ensemble which overcomes individuals' weaknesses. Additionally, we demonstrate how parameters of tree based GP can be diversified to generate multiple models. Finally, we consider whether factoring contributes ensemble members which improve fused accuracy.

- **Cloud-Scale Ensemble Learning**: We develop a cloud scale regression ensemble learning system. Models are generated by multiple cloud-backed virtual machines each supporting an independent parameterized GP learner. Under this scenario we create a fused model (ensemble) on demand at different points in time while the individual GP learners are evolving. The stochasticity of GP and the varying computation speed of virtual machines on the cloud induces variability in the learning rate of each learner. We examine how the performance of the fused models changes in real time. Finally, we discuss treating the convergence of the fusion methods' performance as a stopping criterion for GP.

The remainder of this paper is structured as follows. Section 2 discusses previous and related work. Sections 3 and 4 detail how we generate models and ensembles of models, respectively. Section 5 describes the datasets and parameters of our system used to run the experiments. Finally, Section 6 presents the results of our experiments and Section 7 concludes with some thoughts for future work.

2. RELATED WORK

Collections of multiple models are called "ensembles", whether the models are trained on different subsamples of the data, developed using different parameters of the same learner or simply obtained by different learners pursuing the same prediction task.

2.1 Ensembles in Machine Learning

Over the past two decades numerous researchers in the machine learning community have pursued the idea of generating a great quantity of models for the same data [7,8,22, 25,26]. Similarly, the task of combining predictions from an ensemble of models has attracted attention in recent years. This follows from two observations: there is usually not a single explanation for the data, and multiple models cover the observation space in a more robust manner than a single model can. Initially researchers focused on methods which generated multiple models from the same data irrespective of what kind of learner was being used. These methods relied on repeated subset sampling methods with replacement, e.g. bagging [4] and iterative sampling methods, e.g. boosting [11]. Other examples include random forests [5] and Adaboost. [12]. In this work, we focus on how the changing the parameters of the base learner can generate multiple models.

2.2 GP-based Machine Learning Ensembles

When looking at ensembles built using GP, most of the work has focused on classification [2,10,15,16,23,24]. In classification with GP, models are either built to output discrete class labels or are constructed to yield a continuous number which leads to a class label when converted into class probabilities. Multiple class labels can be fused via majority vote or a sophisticated criterion [19].

Conversely, there has been very little research on regression ensembles [14, 20, 29]. In regression, due to the unconstrained nature of GP models one must perform multiple tests which can guarantee models' outputs are within a reasonable range for any unseen data point before the model is admitted into an ensemble. To build regression ensembles, most current approaches use simple averaging techniques (discussed in Section 4.2). Many examine methods for combining ensembles of parametric models. These methods attempt to understand the differences in the models based on their parameters and/or produce a fused model by fusing the parameters.

3. GENERATION OF MODELS

The availability of massive on-demand computational resources via the cloud enables us to learn many models in parallel. Bagging, boosting or simple parameter variation are now feasible at an unprecedented scale. We use our distributed GP system, FlexGP, to run many instances of GP in parallel in the cloud, thus generating many models [6]. Below we discuss the specifics of the local learner and the factoring settings we used with FlexGP.

3.1 Local learner

Our local learner is based on tree based GP. We built a simple GP system with subtree mutation, single point cross over and tournament selection. We incorporated linear scaling [18,31] to allow GP to learn the shape of the regression function and implemented dynamic operator equalization to control bloating [28]. More parameters of our local learner are listed in Section 5.1.

3.2 Factoring

FlexGP has a probabilistic, user guided means of varying the parameters with which each GP learner executes. It establishes a large scale set of factored learners in a decentralized manner, thus working in parallel to learn a diverse set of models.

We define the parameters of a GP learner running at instance i as $\{L, O, D^i_{tr}, F\}$. L is the operator set provided to GP. O is the objective function used for fitness evaluations in GP. D^i_{tr} is the set of training cases presented to the learner, as defined in Table 2. F is the set of data features used for learning. Different options are available for these parameters and are summarized in Table 1.

Table 1: GP parameters and their possible values and definitions.

Parameter	Value	Definition
Operator Set (L)	W	$\{+, -, /, *\}$
	X	$\{exp, ln\}$
	Y	$\{sqrt, x^2, x^3, x^4\}$
	Z	$\{sin, cos\}$
Objective Function (O)	Norm	Mean absolute error
	Norm-2	Mean squared error
	Norm-inf	Max error
Training Cases (D^i_{tr})	n	Subset of D_{gp}, of size n
Feature Set (F)	m	Subset of features, of size m

Despite the massive number compute resources available to us in the cloud, it is still infeasible to exhaustively try every combination of these factoring settings. Instead, FlexGP assumes a distribution over each of these parameters to probabilistically bias how each instance chooses a value for that parameter when it starts the local GP. For example, one learner could choose $\{W, Norm2, \{d_{1...3000}\}, \{x_1, x_2\}\}$ and another might select $\{W \cup X, Norm2, \{d_{1...3000}\}, \{x_1, x_4\}\}$.

4. COMBINING GP REGRESSION MODELS

As described in Section 2, most previous work which focuses on regression ensembles examines the fusion of parametric models. Since GP produces non-parametric models, we must develop different methods. This requires us to rely on predictions for analyzing the model performance. Each GP learner produces an output $\hat{y}_{mj} = f_m(\overline{\mathbf{x}}_j)$ for each input data point $\overline{\mathbf{x}}_j$, where m and other notational conventions are defined in Table 2. An example of a GP model m is

$$\hat{f}_m(\overline{\mathbf{x}}) = log(x_1) + x_4 x_5 + \frac{x_6}{e^{x_9}}$$

We propose and define six methods for combining ensembles of GP models for regression problems. These methods fall into two distinct categories: selection methods, which select a single model from the ensemble (4.1), and fusion methods, which produce predictions by fusing the predictions from a mixture of models in the ensemble (4.2). Probabilistic methods of fusion are discussed in Section 4.3

Table 2: Problem Notation

Variable	Notation	Definition
Data	D_{gp}	GP training data
	D^i_{tr}	Subset of D_{gp} used by instance i for training
	D^i_{val}	Subset of D_{gp} used by instance i for validation
	D_f	Fusion training data
	D_t	Testing data
Data point	$\overline{\mathbf{x}}_j$	$\overline{\mathbf{x}}_j = \{x_l \mid l = 1...\gamma\}$
Output variable	z_j	$z_j \in \mathbb{R}$, for $\overline{\mathbf{x}}_j$
Model	m	Model m
Prediction	$\hat{y}_{mj} = f_m(\overline{\mathbf{x}}_j)$	Model m's prediction, non-linear in $\overline{\mathbf{x}}_j$
Candidate models	Ω	Set of models for fusion
Predictions for $\overline{\mathbf{x}}_j$	$\overline{\mathbf{y}}_j$	$\overline{\mathbf{y}}_j = \{\hat{y}_{mj} \mid \forall m \in \Omega\}$
Predictions of model m, given D	$\overline{Y}^m_D$	$\overline{Y}^m_D = \{\hat{y}_{mj} \mid \forall \overline{\mathbf{x}}_j \in D\}$
Output estimate	$\hat{z}_j$	ensemble's estimate of z_j

4.1 Selection Methods

Average Model Prediction (AMP)

The simplest selection method is to pick a random model from the ensemble. For each new query, this method would select a model at random and use that to produce a prediction. We measure the effectiveness of this method by measuring the average performance of every model in the ensemble on D_t.

Best A Priori Model (BAM)

As a slight improvement to AMP, we propose BAM: Best A priori Model selection. For predicting the output of new queries, we use the best model from the ensemble based on its MSE on D^i_{val}.

4.2 Fusion Methods

Fusion methods combine multiple models from the ensemble for prediction. They vary in how they mix the models and combine those models' predictions.

Average Ensemble Prediction (AVE)

The simplest fusion method is to generate $\overline{\mathbf{y}}_j$ for a new query $\overline{\mathbf{x}}_j$ and then report the average of those predictions. That is, $\hat{z}_j = \frac{1}{m}\sum_{i=1}^{m} \hat{y}_{ij}$ for each test point $\overline{\mathbf{x}}_j$ [20, 30].

Median Average Model (MAD)

A variant of AVE, MAD finds the *median* prediction among $\overline{\mathbf{y}}_j$ and a prediction for a new query is computed as the average of the prediction of this median model and those of its two neighbors in the prediction space [31].

Adaptive Regression Mixing (ARM)

In Adaptive Regression by Mixing (ARM), each model m is assigned a weight, W_m, which is used to compute $\hat{z}_j$ via a weighted average [33]. The weights are computed in the following manner. Let $r = |D_F|$ and $o = |\Omega|$. First the data D_f is randomly partitioned into two equally sized parts $D^{(1)}$ and $D^{(2)}$. To identify the weights for each model, we assume that the errors for each model are normally distributed. We then use the variance in these errors to identify the weights by executing the following steps:

Step 1: Evaluate σ^2_m which is the maximum likelihood estimate of the variance of the errors, $\overline{e}_m = \{\hat{y}_{mj} - z_j | \overline{\mathbf{x}}_j, z_j \in D^{(1)}\}$. Compute the sum of squared errors on $D^{(2)}$, $\beta_m = \sum_{j=\frac{r}{2}+1}^{r} (\hat{y}_{mj} - z_j)^2$.

Step 2: Estimate the weights using:

$$W_m = \frac{(\sigma_m)^{-r/2} exp(-\sigma_m^{-2}\beta_m/2)}{\sum_{j=1}^{M} (\sigma_j)^{-r/2} exp(-\sigma_j^{-2}\beta_j/2)} \quad (1)$$

Step 3: Redraw subsets $D^{(1)}$ and $D^{(2)}$ and repeat steps 1 and 2. Continue this process for a fixed number of times[1]. Average the weights to get the final weights for the models.

Given a test point $\overline{\mathbf{x}}_j$, predict $\hat{z}_j$ as the weighted average of model predictions: $\hat{z}_j = \sum_{m=1}^{o} W_m \hat{y}_{mj}$.

Transformation for large r: For large values of r, the calculation of the weights as given by equation 1 encounters an underflow error. To avoid this problem we equivalently compute the weights using equations 2 and 3.

$$A_m = -\frac{r}{2} log(\sigma_m) + log(\frac{-{\sigma_m}^{-2}\beta_m}{2}) \quad (2)$$

$$W_m = exp(A_m - log(\sum_{q=1}^{M} A_q)) \quad (3)$$

4.3 Probabilistic Fusion Methods

In meta-model based approaches, given the training data D_f, the joint probability density function $f_{Y,Z}(y, z)$ is built for $\{\overline{\mathbf{y}}_j, z_j | \forall \overline{\mathbf{x}}_j, z_j \in D_f\}$. $f_{Y,Z}(y, z)$ is then used to predict $\hat{z}_j$ for a new input $\overline{\mathbf{x}}_j$ by first computing $\overline{\mathbf{y}}_j$ via the GP generated models and then evaluating the conditional density function for $f_{Z|Y}(\hat{z}_j | \overline{\mathbf{y}}_j)$. An advantage of this approach is the availability of uncertainty information. In our current

[1] An intelligent stopping criteria could be used as well.

contribution, we focus on a non-parametric version of the probabilistic model which does not need any training.

Our model is a non-parametric multivariate kernel density (KDE) estimator, using a gaussian kernel [9, 32]. Let $n = |D_f|$ and $o = |\Omega|$. The multivariate kernel density function is given by

$$f_{Y,Z}(w, v) = \frac{1}{n}\sum_{j=1}^{n}\prod_{m=1}^{o} K_h(w_m - \hat{y}_{mj})K_h(v - z_j)$$

where $K_h(x) = \frac{1}{h}\phi(\frac{x}{h})$, $\phi(x)$ is the standard normal density function and h is the bandwidth. To compute $\hat{z}_k$ for a test point $\overline{\mathbf{x}}_k$, we first compute $\overline{\mathbf{y}}_k$ and then find the expected value of the conditional density function $f_{Z|Y}(z|\overline{\mathbf{y}}_k)$:

$$\begin{aligned}\hat{z}_j &= E(Z|Y = \overline{\mathbf{y}}_k)\\ &= \int_Z z f_{Z|Y}(z|\overline{\mathbf{y}}_k)dz = \int_Z z\frac{f_{Z,Y}(z, \overline{\mathbf{y}}_k)}{f_Y(\overline{\mathbf{y}}_k)}dz\\ &= \frac{\sum_{j=1}^{n}\prod_{m=1}^{o} K_h(\hat{y}_{mk} - \hat{y}_{mj})\int_Z zK_h(z - z_j)dz}{\sum_{j=1}^{n}\prod_{m=1}^{o} K_h(\hat{y}_{mk} - \hat{y}_{mj})\int_Z K_h(z - z_j)dz}\\ &= \frac{\sum_{j=1}^{n} z_j\prod_{m=1}^{o} K_h(\hat{y}_{mk} - \hat{y}_{mj})}{\sum_{j=1}^{n}\prod_{m=1}^{o} K_h(\hat{y}_{mk} - \hat{y}_{mj})}\end{aligned}$$

4.4 Advantages and limitations

Each of the fusion approaches we presented above has certain advantages and limitations. AVE is the simplest of all but could bias the estimation because many correlated models producing similar outputs for a training point. It could also be affected by outliers. MAD, though robust to outliers, ignores a lot more information embedded in more than a few models. Both of these techniques do not differentiate between different models based on their performance on the training data and consider the models themselves to be independent.

ARM presents a unique way of identifying the weights for each model however it can become computationally intensive. In our experiment we resample (according to step 3) 100 times to estimate the weights. The approach is also sensitive to the amount of data presented for training the weights. Once the weights are identified, real time execution of the deployed model is extremely efficient.

ARM, AVE and MAD require an outlier detection algorithm to remove any outliers before fusion. The outlier detection algorithm has to be run in real time. In our approach we estimate the minimum and maximum values for the output variable $z = (z_{min}, z_{max})$ and any model producing predictions that are outside these5C bounds are removed before fusion. For ARM, the weights are renormalized after removing the weights that correspond to the outlier models.

On the other hand, KDE is non-parametric and so is not sensitive to outliers. However the method has two weaknesses. First, the method requires the user to set the bandwidth parameter for the kernel. The performance of the algorithm is sensitive to this parameter and setting this value might require training. In this contribution we choose a bandwidth parameter heuristically. Second, the method requires us to carry the data to deployment scenario, and as the predictions are made in real time, this data is queried to make the calculations. This makes predicting outputs for new input computationally expensive.

5. EXPERIMENTAL SETUP

To demonstrate the efficacy of GP regression ensembles we experimented with 2 datasets, summarized in Table 3. In the first dataset, a number of settings in a power plant are used to predict the NOx emissions [3]. These settings (features) are continuous-valued and are known to have a non-linear relation with the NOx emissions. We abbreviate this problem as NOx.

Table 3: Summary of dataset statistics.

Dataset	$\lvert D_{gp}\rvert$	$\lvert D_f\rvert$	$\lvert D_t\rvert$	Total	$\lvert\overline{\mathbf{x}}_j\rvert$	Range of z
NOx	4,017	310	900	5,227	18	[0.270 0.654]
MSD	398,924	67204	49436	515564	90	[1922 2011]

Our second problem comes from the *music information retrieval* (MIR) community where a dataset of one million song tracks has been created to push the focus towards solving large scale MIR problems [1]. We set out to study the *year prediction problem*, where the task is to predict the release year of a song, given a set of track features. Of the million tracks in the dataset, roughly half have year data. We abbreviate this problem as MSD.

5.1 GP and FlexGP settings

Our experiments use a GP based learner described in Section 6. We use the same GP configuration as described by Koza with select differences [21]. The learner is set with a population of size 1000. The mutation rate is 0.1, the crossover rate is 0.6, and the max depth is set to 12 for the NOx experiments and is set to 8 in the MSD experiments. The tournament selection size is 7 and we used ramped-half-and-half as the algorithm for tree growth. The learners were allowed to run until stopped in the NOx experiments, but were terminated after 20 generations in the MSD experiments. Our method for advancing to the next generation is Silva's dynamic equalization with a bin width of 5 [28]. We ran multiple experiments with each dataset, each time with different amounts of training data or GP learner settings. Table 4 presents the settings for different runs.

Table 4: Settings of different experiments

Dataset	Exp.	$\frac{\lvert D^i_{tr}\rvert}{\lvert D_{gp}\rvert}$	# Nodes	Parameter Factored
NOx	1	67%	150	None
	2	67%	150	F
	3	67%	150	D
	4	67%	150	D & L
	5	67%	150	D & O
MSD	1	10%	75	ALL
	2	1%	150	ALL
	3	0.1%	150	ALL

For every experiment, except MSD1, we use 150 virtual machines, each with a single core and 2GB of RAM. In MSD1, each node required 3.4 GB of memory to load D^{tr}_{gp} into Java. Thus each node ran a virtual machine with two cores and 4GB of RAM, which forced us to halve the number of virtual machines used (we are core-limited).

Due to the variance in starting times across our nodes, each node runs for a different number of generations (up to a total of 20 generations in the MSD experiments). We collect the best model for each generation from each node. On average, we obtain about 30 models (20 models in the

Table 5: MSE and PG_{MSE} on the NOx experiments for all fusion methods. Note that the MSE value for AMP on NOx1 is used for the baseline in PG_{mse} calculations in all NOx experiments.

		AMP	BAM	AVE	MAD	ARM	KDE
NOx1	MSE	1.97E-3	1.37E-3	1.42E-3	1.65E-3	1.30E-3	6.63E-4
	PG_{MSE}	0.0%	30.4%	28.0%	16.4%	34.2%	66.4%
NOx2	MSE	2.35E-3	1.33E-3	1.54E-3	1.92E-3	1.29E-3	5.60E-4
	PG_{MSE}	-19.2%	32.4%	22.2%	2.8%	34.6%	71.6%
NOx3	MSE	2.10E-3	1.38E-3	1.41E-3	1.43E-3	1.31E-3	6.40E-4
	PG_{MSE}	-6.4%	30.0%	28.6%	27.5%	33.5%	67.6%
NOx4	MSE	1.66E-3	1.51E-3	1.26E-3	1.27E-3	1.22E-3	6.82E-4
	PG_{MSE}	15.8%	23.7%	36.0%	35.5%	38.2%	65.5%
NOx5	MSE	2.15E-3	1.30E-3	1.44E-3	1.54E-3	1.29E-3	6.53E-4
	PG_{MSE}	-9.0%	33.9%	26.8%	21.7%	34.6%	66.9%

MSD experiments) from each node giving us approximately 5000 models in the NOx experiments and around 2000-3000 models for the MSD experiments.

5.2 Data splits

We divide each dataset into three parts: for training GP models (D_{gp}), for training the fusion model (D_f) and for testing (D_t). Within D_{gp}, each node selects a random subset for training the GP models (D^i_{tr}) and reserves the rest for validation (D^i_{val}). Tables 3 and 4 report the size of these splits for both datasets.

For MSD data, the MIR commmunity has established a standardized train/test split to facilitate comparisons across different approaches [1]. This split ensures that no artist is present in both subsets. In this paper, we further split the provided MIR training dataset to produce D_{gp} and D_f. When constructing this split and when each node splits D_{gp} into D^i_{tr} and D^i_{val}, we continue to ensure that no artists appear in more than one subset.

6. RESULTS AND DISCUSSION

To measure the performance of our fusion strategies, we compute their MSE on D_t. To aid in comparing the performance of these strategies, we need to construct a comparative performance measurement. To do this, we establish the performance of a single method on a single experiment within each problem as a baseline performance value, $baseline_{MSE}$, which all other methods are measured against. Given the performance of a method in an experiment, $method_{MSE}$, we compute its *performance gain* as

$$PG_{MSE} = \frac{baseline_{MSE} - method_{MSE}}{baseline_{MSE}}$$

This number is positive if we achieved a better performance and negative otherwise.

For the NOx experiments, we set our basline as the performance of AMP in the NOx1 experiment. Because we executed NOx1 without any factorings, we ran an identical learner with the same data in parallel at each instance. This corresponds to what a standard run of classic GP would yield. Which is a good baseline for the NOx experiments, where we are concerned with how factoring affects fusion performance.

Our focus with the MSD experiments is the effects of the size of D^i_{tr} (see Section 6.3) on ensemble performance. To provide a similar measure of performance gain for the MSD experiments, we set the baseline to be the MSE of AMP on D_t for the MSD1 experiment. This baseline effectively highlights the impact of decreasing training set size on fusion performance.

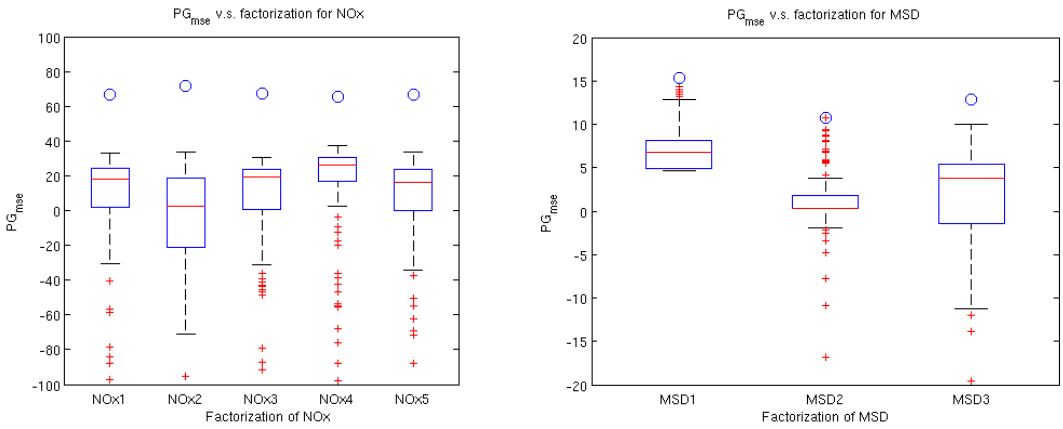
Table 6: MSE and PG_{MSE} on the MSD experiments for all fusion methods, except KDE. The MSE value for AMP from MSD1 is used as the baseline for the computation of each PG_{MSE}

		AMP	BAM	AVE	MAD	ARM
MSD1	MSE	122.7	105.2	111.6	111.7	103.9
	PG_{MSE}	0.0%	14.3%	9.0%	9.0%	15.4%
MSD2	MSE	117.3	106.5	112.6	113.4	104.7
	PG_{MSE}	4.4%	13.2%	8.2%	7.6%	14.7%
MSD3	MSE	123.6	111.4	109.7	114.4	107.8
	PG_{MSE}	-0.7%	9.2%	10.6%	6.8%	12.1%

We present the results of implementing the fusion strategies from Section 4 to the NOx and MSD problems in Tables 5 and 6, respectively. There are no results for KDE in Table 6 because we could not compute KDE due to the MSD dataset size.

In Figure 1, we have computed the PG_{MSE} for each model considered for fusion in each experiment from Table 4 and plotted the resulting distribution as a box plot. For each box, the central line is the median of the distribution and the edges mark the 25^{th} and 75^{th} percentiles. The whiskers extend to the most extreme points which are not considered as outliers, and all outliers are marked by an 'x'. The circles indicate the highest performance gain achieved by any fusion strategy for each experiment.

We spend the next several paragraphs investigating a variety of questions. First, we examine the performance improvements which could be achieved by generating and fusing an ensemble of models. We also examine the efficacy of various problem factorizations. Next, we consider the performance of fusion with a relatively small sample of training data at each instance. This allows the minimization of both the overall GP training time and the memory requirements of each individual learner. Finally, we examine the performance of our system's predictions in real-time, and we detail the behavior and characteristics of several fusion strategies when considered in this setting.



(a) NOx (b) MSD

Figure 1: The quartile distribution of PG_{MSE} of models used for fusion in each experiment. The circles represent the best PG_{MSE} from fusion. *Left* Results for NOx experiments; KDE was the best fusion method. *Right* Results for MSD experiments; ARM was the best fusion method.

6.1 Performance of Fusion Strategies

For the NOx dataset, KDE performs the best in all trials. ARM provides the second best performance gains. Selecting the best model among the ensemble based on their performance provides reasonably good performance gains. To our surprise MAD, which was most commonly used in previous GP-based regression ensemble work, did not perform well. This is because median average utilizes very few predictions around the median, thus ignoring the bulk of the predictions.

For MSD experiments the three factorings simply reduce the amount of the data used for training. We generally see the performance gain values fall as less data is provided to the GP learners. The exception is the AVE ensemble performance. Interestingly, AVE produces the best results MSD3, when each learner is given only 400 training points. We hypothesize this is because as data size is reduced to such small size each learner specializes in a different region of the input space and makes different errors. Thus, the models have uncorrelated errors and taking the average prediction becomes a good strategy for combining models [25].

6.2 Impact of Different Factorings

With the NOx experiments, we examine the impact of different factorings on ensemble performance. Looking at the NOx results in Figure 1, we notice models learned in the NOx4 experiment are generally better and the distribution is tighter than the rest (although there is a long tail of outliers). Conversely, we see that the models from NOx2 have less consistent results and perform worse than the rest. This is understandable since NOx2 is built with subset of features.

We first examine the performance gain of the AMP method across experiments. AMP results in worse performance than the classic GP baseline in all experiments except NOx4. This suggests that factoring the operator set (L) could be the linchpin in seeing performance gains from fusion. However, this merits further study as this result isn't observed across other the other fusion strategies.

Finally, we note that KDE and ARM result in consistently good performance gains, irrespective of factoring. This suggests that factor settings may not have such a large impact on ensemble performance.

6.3 Reducing the Size of the Data

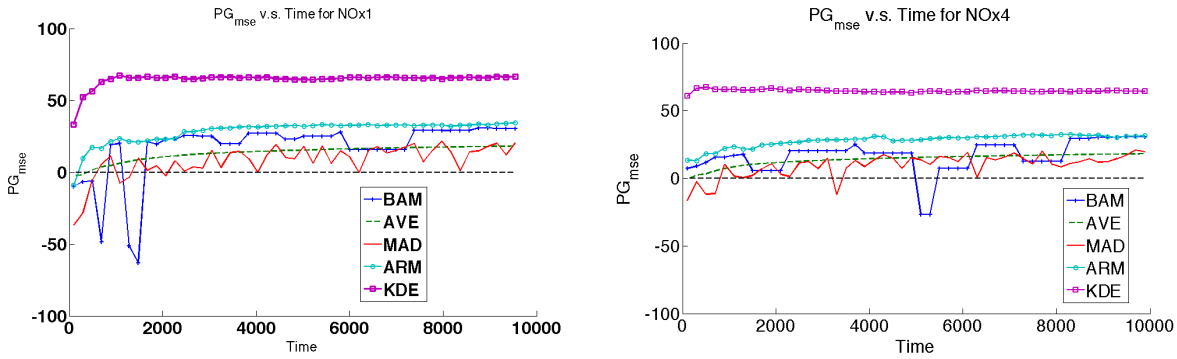
In our experiments we examine how reducing the size of the data used by each local learner affects ensemble performance. Reducing the training data directly decreases the time taken to evaluate an individual, allowing for faster learning overall. However, learning on less data typically comes at the cost of decreased performance.

When decreasing the size of data in the MSD experiments, we see that while individual model performance drops, the fusion results remain better than baseline. As reported in Table 3, we experimented with using 10%, 1%, and 0.1% (40K, 4K and 400 data points, respectively) for training. As we can see in Figure 1 right, the resulting individual models perform less reliably as the number of training cases decreases. This is shown by the increased variance in the model performance. However, as reported in Table 6 the ensemble performance shows only slight decrease in performance. Indeed, ARM fusion performs better than the baseline in all cases. This demonstrates that decreasing the amount of training data used by a factor of 100 still allows for improved performance of the ensemble.

6.4 Analyzing Real-Time Fusion

In this section we assess how the performance of the fusion strategiess perform as time progresses and the GP models evolve more generations. Figure 2 presents the performance of the different fusion strategies in real time on the NOx problem. At each time step we collect the models so far, construct ensembles using the proposed methods, compute their predictions on D_t and report their performance gain. Plots for NOx2, NOx3 and NOx5 were omitted as they are highly similar to the three shown.

The plots in Figure 2 illustrate a couple interesting points. First, notice BAM is not robust and can give arbitrarily poor performance at different points in time. This is because BAM might select a model which has overfit on D^i_{tr} in such a way that it still does well on D^i_{val} but performs poorly



(a) NOx1 (b) NOx4

Figure 2: Improvements achieved over the baseline (no fusion) methodology as generations are executed and time (in seconds) progresses in our framework for NOx data.

on D_t. Thus, even though it achieves similar performance gains to ARM at some points, it is not a reliable strategy. Second, notice that ARM, KDE and AVE have stable and incremental performance. That is, as time progresses they either improve or remain at their last attained performance. This sort of performance stabilization over time suggests we could use the fused performance as a stopping criteria for the entire system.

7. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a regression ensemble system which produces model ensembles by relying on a GP-based learner. Our system pairs the stochastic nature of GP with a probabilistic parameter factoring method and the variable computation speed of the cloud to produce many diverse models for fusion. We presented multiple robust strategies for fusing predictions from these ensembles of models. We also explored the affect of this probabilistic factoring method on the generation of model ensembles. A discussion of other results and suggested follow-on work follows.

Informed Factoring Distributions In our current implementation, we set all distributions for factoring to be static, uniform distributions. However, since these distributions are retrieved from the parent node, they could change dynamically as new nodes are launched. For example, a node could modify $p(D^i_{tr})$ to decrease the chance any node it starts will select the same training cases it did. Alternatively, if new nodes are to be brought up after the system has been running, $p(D^i_{tr})$ could be modified to weight difficult training cases more and $p(F)$ could be modified to weight features with little apparent informative power less. We hope to explore such possibilities in the future.

Data Challenge: We challenged ourselves to give each learner as little data as possible while still retrieving fused performance equivalent to the case where multiple learners are given large amounts of data. This enables our iterative learning mechanism to learn quickly from large data. We have been particularly successful in this challenge. For one of our problems we have provided as little as 400 training data points from a total of 400,000 available points. Under these constraints we achieved comparable performance to the case where 40,000 data points were made available. We also achieved comparable performance to the benchmark available in literature for this dataset, which utilises all data for training. As part of the future work we intend to further explore this aspect of our system and to seek answers to the following questions: How much data at each learner is enough? How many models, where each is trained on a small subset, are enough?

Repeating experiments While the results presenting are compelling, little attention has been paid to how robust they are. Ensembles are built from thousands of models, which should be sufficiently large enough to ensure we have used a representative sampling of possible models. However, a more rigorous and extensive experimental design incorporating cross-validation and repeated trials is needed before any deeper conclusions can be drawn.

Multiple Fusion Approaches: We have chartered the largely unexplored territory of how to fuse multiple continuous-valued predictions of regression ensembles. We presented multiple strategies with different complexities. We show conventionally used methods of average and median average do not yield good results.

Ensemble Learning at Scale: Finally, we presented a system which factors the data and the genetic programming learner in multiple ways. We demonstrated our system's capability to provide accurate intermediate fused answers as time progresses. As part of our future work we intend to demonstrate our system's capacity to leverage the power of thousands of nodes by testing the system on larger datasets where many learners build models independently.

Acknowledgements

This work was supported by the GE Global Research center. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of General Electric Company.

8. REFERENCES

[1] T. Bertin-Mahieux, D. Ellis, B. Whitman, and P. Lamere. The million song dataset. In *ISMIR 2011: Proceedings of the 12th International Society for Music Information Retrieval Conference, October 24-28, 2011, Miami, Florida*, pages 591–596. University of Miami, 2011.

[2] U. Bhowan, M. Johnston, M. Zhang, and X. Yao. Evolving diverse ensembles using genetic programming

for classification with unbalanced data. *Evolutionary Computation, IEEE Transactions on*, 2012.

[3] P. Bonissone. Lazy meta-learning: creating customized model ensembles on demand. *Advances in Computational Intelligence*, pages 1–23, 2012.

[4] L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[5] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[6] O. Derby, K. Veeramachaneni, and U.-M. OâĂŹReilly. Cloud driven design of a distributed genetic programming platform. In A. Esparcia-AlcÃązar, editor, *Applications of Evolutionary Computation*, volume 7835 of *Lecture Notes in Computer Science*, pages 509–518. Springer Berlin Heidelberg, 2013.

[7] T. Dietterich. Ensemble methods in machine learning. *Multiple classifier systems*, pages 1–15, 2000.

[8] T. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine learning*, 40(2):139–157, 2000.

[9] V. A. Epanechnikov. Non-parametric estimation of a multivariate probability density. *Theory of Probability & Its Applications*, 14(1):153–158, 1969.

[10] G. Folino, C. Pizzuti, and G. Spezzano. Mining distributed evolving data streams using fractal gp ensembles. *Genetic Programming*, pages 160–169, 2007.

[11] Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *Machine learning international conference*, pages 148–156. Morgan Kauffman Publishers, Inc., 1996.

[12] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[13] M. Friese, O. Flasch, K. Vladislavleva, T. Bartz-Beielstein, O. Mersmann, B. Naujoks, J. Stork, and M. Zaefferer. Ensemble-based model selection for smart metering data. In *Proceedings 22 Workshop Computational Intelligence*, page 215, dec 2012.

[14] H. Iba. Bagging, boosting, and bloating in genetic programming. In *Proceedings of the genetic and evolutionary computation conference*, volume 2, pages 1053–1060, 1999.

[15] K. Imamura, T. Soule, R. Heckendorn, and J. Foster. Behavioral diversity and a probabilistically optimal gp ensemble. *Genetic Programming and Evolvable Machines*, 4(3):235–253, 2003.

[16] U. Johansson, T. Löfström, R. König, and L. Niklasson. Genetically evolved trees representing ensembles. *Artificial Intelligence and Soft Computing–ICAISC 2006*, pages 613–622, 2006.

[17] A. J. Keane, A. Choudhury, P. B. Nair, A. J. K. F. Choudhury, and P. Nair. A data parallel approach for large-scale gaussian process modeling. In *in Proc. the Second SIAM International Conference on Data Mining*. Citeseer, 2002.

[18] M. Keijzer. Improving symbolic regression with interval arithmetic and linear scaling. In C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, and E. Costa, editors, *Genetic Programming*, volume 2610 of *Lecture Notes in Computer Science*, pages 275–299. Springer Berlin / Heidelberg, 2003. 10.1007/3-540-36599-0_7.

[19] J. Kittler, M. Hatef, R. Duin, and J. Matas. On combining classifiers. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(3):226–239, 1998.

[20] M. Kotanchek, G. Smits, and E. Vladislavleva. Trustable symbolic regression models. *Genetic Programming Theory and Practice V, Genetic and Evolutionary Computation*, pages 203–222, 2007.

[21] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[22] A. Krogh and J. Vedelsby. Neural network ensembles, cross validation, and active learning. *Advances in neural information processing systems*, pages 231–238, 1995.

[23] W. Langdon, S. Barrett, and B. Buxton. Combining decision trees and neural networks for drug discovery. *Genetic Programming*, pages 76–89, 2002.

[24] P. L. Lanzi. XCS with stack-based genetic programming. In R. Sarker, R. Reynolds, H. Abbass, K. C. Tan, B. McKay, D. Essam, and T. Gedeon, editors, *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003*, pages 1186–1191, Canberra, 8-12 Dec. 2003. IEEE Press.

[25] M. Perrone and L. Cooper. When networks disagree: Ensemble methods for hybrid neural networks. Technical report, DTIC Document, 1992.

[26] J. Quinlan. Bagging, boosting, and c4. 5. In *Proceedings of the National Conference on Artificial Intelligence*, pages 725–730, 1996.

[27] M. Schmidt and H. Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009.

[28] S. Silva. Handling bloat in GP. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, GECCO '11, pages 1481–1508, New York, NY, USA, 2011. ACM.

[29] K. Veeramachaneni, K. Vladislavleva, M. Burland, J. Parcon, and U. O'Reilly. Evolutionary optimization of flavors. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 1291–1298. ACM, 2010.

[30] K. Veeramachaneni, K. Vladislavleva, M. Burland, J. Parcon, and U. M. O'Reilly. Evolutionary optimization of flavors. In J. Branke and et al, editors, *GECCO '10: Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 1291–1298, Portland, Oregon, USA, 7-11 July 2010. ACM.

[31] C. Vladislavleva and G. Smits. Symbolic regression via genetic programming. *Final Thesis for Dow Benelux BV*, 2005.

[32] M. P. Wand and M. C. Jones. *Kernel smoothing*, volume 60. Chapman & Hall/CRC, 1995.

[33] Y. Yang. Regression with multiple candidate models: selecting or mixing? *Statistica Sinica*, 13(3):783–810, 2003.