Simultaneous Ensemble Generation and Hyperparameter Optimization for Regression

David Roschewitz, Kurt Driessens, Pieter Collins

Maastricht University, P.O. Box 616 6200 MD Maastricht, The Netherlands,
d.roschewitz@student.maastrichtuniversity.nl

Abstract. The development of advanced hyperparameter optimization algorithms, using e.g. Bayesian optimization, has encouraged a departure from hand-tuning. Primarily, this trend is observed for classification tasks while regression has received less attention. In this paper, we devise a method for simultaneously tuning hyperparameters and generating an ensemble, by explicitly optimizing parameters in an ensemble context. Techniques traditionally used for classification are adapted to suit regression problems and we investigate the use of more robust loss functions. Furthermore, we propose methods for dynamically establishing the size of an ensemble and for weighting the individual models. The performance is evaluated using three base-learners and 16 datasets. We show that our algorithms consistently outperform single optimized models and can outperform or match the performance of state of the art ensemble generation techniques.

Keywords: Bayesian optimization, hyperparameter optimization, ensemble generation, regression

1 Introduction

Hyperparameter tuning *for regression* is sparsely covered in research and its combination with ensemble generation appears to be entirely absent. This although both techniques have been succesfully applied to classification problems. Our research addresses this omission by examining methods to automatically generate ensembles with tuned hyperparameters for regression problems.

Naïve search methods have commonly been used for hyperparameter optimization of machine learning models. Grid search, for instance, evaluates hyperparameters on a grid with a predefined resolution. This, and its inefficiency in high-dimensional space, limits the usefulness for practical applications. Random search [2] partially alleviates this limitation, but unlike Bayesian optimization, it does not leverage all available information in the tuning process [22]. Researchers have expressed a need to explore both regression problems and deep learning in the context of hyperparameter optimization to encourage the departure from hand-tuning [3, 9].

It is well accepted that ensemble methods, which consist of a combination of multiple base-learners, generally outperform single models for many types of

problems [19]. Therefore, combining ensemble generation with hyperparameter tuning is a natural step for self-optimizing algorithms. Overproduce-and-select (OPAS) methods are a popular choice for ensemble construction, and can achieve best-in-class performance [6]. During an OPAS procedure, the ensemble is constructed selecting learned models from a library of predictors, which could be the models evaluated during a hyperparameter optimization procedure.

Lévesque et al. [15] devised a method that optimizes hyperparameters and simultaneously constructs the ensemble for classification problems. Essentially, it uses all previously trained models in the context of the ensemble for the Bayesian optimization step, which computes the hyperparameters of good base-models to add to the ensemble. They demonstrate that this way of generating ensembles can outperform OPAS methods, with no significant increase in runtime, and the benefit of not requiring a library of trained models.

In their investigation, Lévesque et al. showed that the optimization and ensemble generation step can be effectively combined. Their research, however, does not analyze the suitability of the method for regression problems, which require different treatment. Furthermore, critical elements such as ensemble size and the combination function of the ensemble were fixed. This paper will discuss and develop algorithms based on the research of Lévesque et al. [15] and releases the constraint of a fixed ensemble size. 16 small and medium sized publicly available data sets are used to evaluate the proposed solutions.

The paper is structured as follows: Section 2 formally introduces Bayesian hyperparameter optimization and ensemble generation approaches. Section 3 outlines our contributions, specifically modifications and extensions made to the algorithm. Section 4 presents the experimental set-up, results and analysis.

2 Hyperparameter Optimization

Hyperparameter optimization minimizes a function $f(\gamma)$, where γ is a set of hyperparameters in Γ, the hyperparameter space. In this context, $f(\gamma)$ can be evaluated by training a model with parameters γ, $M(\gamma)$, and computing its performance. This can be measured using a so-called loss function L. Previous observations, $\mathcal{D}$, of the performance of parameters γ can then be used for various optimization algorithms.

The following sections explain how Bayesian optimization is used to optimize $f(\gamma)$ and how ensembles can be generated from a set of trained models. Lastly the combined optimization and ensemble generation procedure, on which this research is based, is presented.

2.1 Bayesian Optimization

In contrast to naïve search methods such as grid or random search, Bayesian optimization uses *all* previous observations $\mathcal{D}$ to create a probabilistic model, sometimes called a *surrogate function*, $\hat{f}(\gamma)$ of the objective $f(\gamma)$. A so-called

acquisition function then computes the next point in Γ to evaluate. The model is updated with the performance of this point, and the two steps are repeated.

The surrogate function is the *posterior* distribution over the space of functions, induced by the *prior* and our observations $\mathcal{D}$. The prior captures our beliefs about the space of possible objective functions. For a more complete review of BO see, e.g., Brochu et al. [4] or for a broader overview see, e.g., Shahriari et al. [21].

Gaussian Process Prior Gaussian process (GP) priors are a common choice for Bayesian optimization, due to their flexibility and tractability. A GP is a distribution over functions of type $f : \Gamma \to \mathbb{R}$, and when combined with observations $\mathcal{D}$ induces a posterior over functions. GPs are defined fully by their mean function $m : \Gamma \to \mathbb{R}$ and covariance function $k : \Gamma \times \Gamma \to \mathbb{R}$.

For hyperparameter optimization, the use of an automatic relevance detection (ARD) Matérn 5/2 kernel is suggested in literature [22].

$$\begin{aligned} k_{\mathrm{M52}}(x, x') = \theta_0 \exp(-\sqrt{5r^2(x, x')}) \\ (1 + \sqrt{5r^2(x, x')} + \frac{5}{3}r^2(x, x')). \end{aligned} \tag{1}$$

$$r^2(x, x') = \sum_{d=1}^{D} (x_d - x'_d)^2 / \theta_d^2. \tag{2}$$

The use of ARD kernels typically results in a total of $D+3$ GP parameters[1], θ: kernel length-scales $\theta_{1:D}$, kernel amplitude θ_0, observation noise v and constant mean m [22].

Acquisition Function By construction of the GP prior, the surrogate $\hat{f}(\gamma)$ has both a predictive mean function $\mu(\gamma)$ and predictive variance function $\sigma(\gamma)$. The acquisition function $a : \Gamma \to \mathbb{R}^+$ can then be used to determine the *utility* of any point in Γ. The next set of hyperparameters to evaluate are then simply computed as $\gamma_{n+1} = \mathrm{argmax}_\gamma \; a(\gamma)$. Snoek et al. suggest using the Expected Improvement (EI) acquisition function, which provides a tradeoff between exploitation and exploration [22]:

$$a_{EI}(\gamma) = \sigma(\gamma)(\alpha(\gamma)\Phi(\alpha(\gamma)) + \phi(\alpha(\gamma))) \tag{3}$$

$$\alpha(\gamma) = \frac{f(\gamma_{\mathrm{best}}) - \mu(\gamma)}{\sigma(\gamma)} \tag{4}$$

where $\Phi(x)$ is the CDF and $\phi(x)$ the PDF of the standard normal distribution.

Since the mean and variance functions are dependent on the GP parameters θ, they can be represented as $\mu(\gamma; \theta)$ and $\sigma(\gamma; \theta)$. For the fully Bayesian treatment

[1] We employ the term 'GP parameters' to emphasize the difference between these and the hyperparameters subject to optimization in this paper.

of the GP parameters a so-called integrated acquisition function will be used throughout this paper. It is computed through a Monte Carlo estimate, using slice sampling for efficient computation of the required samples. See [22] and their additional material for further details on how the acquisition function is estimated.

2.2 Ensemble Construction from Optimization Output

Post-hoc ensemble generation (PHEG) is a natural way of constructing an ensemble from a library of trained models. PHEG is the *selection* stage of an OPAS method. In the context of optimizing hyperparameters, the history of all trained models serves as input for the ensemble generation.

The PHEG procedure works well in practice using a greedy selection criteria, which can be based on a variety of performance measures such as e.g., mean-squared-error (MSE) [6]. The procedure works as follows:

1. Begin with an empty ensemble.
2. Select the model which maximizes the ensemble performance.
3. Repeat 2 for a given number of iterations, until all models are added or until a point of diminishing returns is reached.

Caruana et al. [6] suggest modifying the procedure to allow selection with replacement. A noteworthy advantage of PHEG procedures, is that it can utilize the same performance function as the final benchmark, irrespective of how the library of models was acquired. When comparing our proposed algorithms to PHEG, we employ selection with replacement.

2.3 Simultaneous Ensemble Generation and Optimization

Hyperparameter optimization and ensemble construction are typically treated as separate procedures, with the possibility of coupling the techniques. Lacoste et al. [14] propose a bootstrapped round-robin technique, where each model of a fixed ensemble is optimized independently. Fundamentally, their ensemble sequential model-based optimization (ESMBO) procedure allows for more computationally efficient model optimization, but uses no information about the ensemble performance in the optimization.

In 2016 Lévesque et al. [15] outlined a simultaneous ensemble generation and optimization approach (SEGO). SEGO considers the performance of the ensemble, E, at every iteration. The loss function, introduced earlier, can be reformulated to evaluate the ensemble, not a single model.

Optimizing the hyperparameters of every model in E would make the objective space excessively high dimensional for large ensemble sizes. An elegant solution used by SEGO is to let the objective function $f(\gamma|E)$ be the loss of the ensemble if a model m trained with parameters γ is placed at index j of E. Hence, the *value* of parameters γ is measured *given* the current ensemble E.

$$f(\gamma|E, j) = L(E[j] \leftarrow M(\gamma)) \tag{5}$$

Throughout the SEGO algorithm, all trained models and their hyperparameter are stored in a history H and P respectively. The round-robin procedure optimizes the model at index $j = i \bmod n$ at iteration i and for ensemble size n. The loss of replacing $E[j]$ with every model in H is computed as l, using Formula 5. The loss of each hyperparameter in P can now be represented by l, and the two lists serve as input to the Bayesian opimization step, which then constructs the surrogate model $\hat{f}(\gamma)$.

The consequence of this type of loss calculation, is that for every iteration the loss must be computed $|H|$ times, resulting in runtime of $O(|H|m)$ for m prediction instances. This overhead is tolerated as the GP computation of the surrogate model contains a matrix inversion, which runs in $O(|H|^3)$. In addition, the assumption that training a learning algorithm is significantly more costly holds in practice.

Algorithm 1 formalizes the SEGO algorithm, but is adjusted to follow the notation of this paper. Note the explicit notion of the cross-validated loss computation, where the loss is averaged. In our research, the loss is estimated through 5-fold cross-validation. We employ *Spearmint* provided by Snoek et al. [22][2] as the implementation of Bayesian optimization for hyperparameter tuning. Constant Gaussian noise is assumed.

Algorithm 1: Simultaneous Ensemble Generation and Optimization

Input : b, n, $a()$, $L()$, $M()$
Output: Ensemble E; history of models H

$E, H, P \leftarrow \emptyset$;
for $i \leftarrow 1$ **to** b **do**
  $j \leftarrow i \bmod n$;
  $l \leftarrow \text{cross-val}([L(E[j] \leftarrow m)]_{m \subset H})$;
  $\hat{f}(\gamma) \leftarrow \text{BO}(P, l)$;
  $\gamma_i \leftarrow \text{argmax } a_{EI}(\gamma; \hat{f}(\gamma))$;
  $m_i \leftarrow M(\gamma_i)$;
  $l \leftarrow l \cup \{\text{cross-val}(L(E[j] \leftarrow m_i))\}$;
  $H \leftarrow H \cup \{m_i\}$;
  $P \leftarrow P \cup \{\gamma_i\}$;
  $E[j] \leftarrow H[\text{argmin}_k l[k]]$;
end

The research performed by Lévesque et al. showed promising results: Their algorithm outperforms a single optimized model. In some cases SEGO also performed better than an ensemble constructed post-hoc from all models trained

[2] Code available at
https://github.com/JasperSnoek/spearmint

throughout the procedure [15]. If a loss function different from the final ensemble evaluation is used for the optimization, they suggest using the PHEG method on the history H from the SEGO procedure.

SEGO was designed and tested for classification tasks only, an omission our research addresses. The authors also note that the requirement of a fixed ensemble size n should be investigated. Furthermore, we find that different loss functions as well as ensemble weighting functions should be explored. This could both improve the algorithm and also demonstrate its robustness. In the following section we introduce SEGO for regression, suitable loss functions, dynamic sizing approaches and a non-constant weighting technique.

3 SEGO for Regression

In order to apply the SEGO procedure to regression problems, the ensemble *integration function* (sometimes referred to as weighting or combination function) and the loss function must be chosen. A default choice for ensembles, is to average the results of all models. The ensemble prediction is therefore a linear combination of all model predictions, with weights $w_{1:n} = \frac{1}{n}$ for ensemble size n.

The residuals of a predictor can be defined as $r_i = \hat{y}_i - y_i$. A typical loss function for regression problems is the mean-squared-error (MSE): $MSE = \frac{1}{n}\sum_{i=1}^{n}(r_i)^2$. With integration and loss function defined, the baseline SEGO for regression (SEGOR) is established.

3.1 Loss function

The fashion with which the loss of parameters given an ensemble, $f(\gamma|E)$, is computed is instrumental to the iterative optimization of the ensemble. Firstly, the cross-validated loss is used directly to select the next hyperparameters γ_{i+1}, and the loss is also used to choose which model to place at index j.

A known drawback of MSE is that it places disproportionately high weights on large errors. In the context of SEGOR this means a model might be selected if it reduces the error on a difficult-to-predict outlier, but in fact reduces the performance on most unseen instances.

Regularization techniques aim to reduce overfitting, usually by complementing the minimization function with a *regularization term*. Since we are not performing a straight-forward minimization, but using an iterative procedure, adding such a term is non-trivial. However, there exist robust loss functions that aim to reduce overfitting.

One such function is the *Huber loss*, a type of robust M-estimator. In 1964 Huber introduced the idea of a generalized M-estimator minimizing $\sum_{i=1}^{n}\rho(x_i)$ where ρ is some function on data x [11]. Huber loss is defined as a piecewise

function of residuals whose behaviour is governed by $|r|$. This significantly reduces the impact of large errors, reducing the likelihood to overfit.

$$L_{\text{huber}}(r) = \begin{cases} \frac{1}{2}r^2 & \text{if } |r| < c \\ c(|r| - \frac{1}{2}c) & \text{otherwise} \end{cases}$$

Another, more extreme, type of robust loss function is the *Tukey bisquare*. In contrast to Huber loss, for large $|r|$, the loss is constant. This further reduces the weight given to large errors. Tukey's bisquare has been used successfully as a loss function for other regression problems [1].

$$L_{\text{tukey}}(r) = \begin{cases} \frac{c^2}{6}[1 - (1 - (\frac{r}{c})^2)^3] & \text{if } |r| < c \\ \frac{c^2}{6} & \text{otherwise} \end{cases}$$

The chosen values of c for Huber loss and Tukey's bisquare are $c = 1.345$ and $c = 4.685$, respectively. At these values of c, both loss functions are at least 95% as accurate as least-squares if the data is sampled from a normal distribution. Figure 1 highlights the differences in the loss functions.

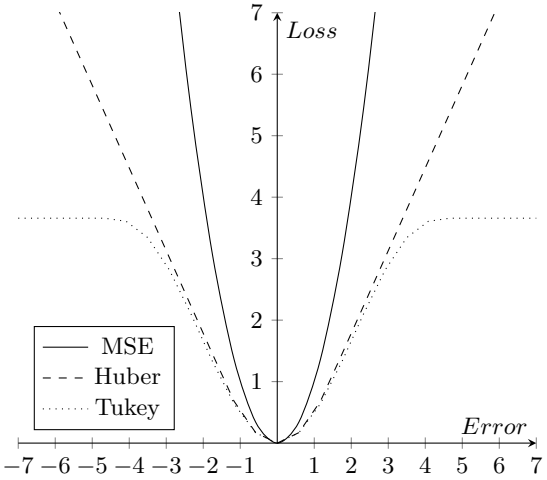

Fig. 1: Comparison of loss functions

3.2 Ensemble size

Much of ensemble research has focused on selecting an ensemble from a library of models [5, 12]. SEGOR currently optimizes parameters given a fixed ensemble size n. Such ensemble generating algorithms are sufficiently novel that there are no established approaches to dynamically adjust the ensemble size throughout the optimization procedure.

Lévesque et al. empirically selected $n = 12$ for their investigation, and our work confirms that this is a reasonable choice, but this might not hold for all types of base-models and datasets. We therefore devised multiple methods of dynamically generating the ensemble *throughout* the optimization process with *no* fixed size.

The fixed ensemble size method (F) will serve as the baseline. Note that the modifications to non-fixed ensemble sizes are performed *after* the BO step, and hence do not affect the input of the hyperparameter tuning. The first proposed method, *best-growing* (BG), initializes the ensemble as $E = \emptyset$, and can add any model $m \in H$, in addition to replacing the model at index j. If a model is added to the ensemble, the remainder of the ensemble is left unchanged. This means that at iteration i, i models are evaluated at index j, and i models are evaluated as additions. An additional set of losses, l_{grow} is computed:

$$l_{\text{grow}} = \text{cross-val}([L(E \cup m)]_{m \subset H}) \tag{6}$$

Similarly, the *best-dynamic* (BD) method allows for the deletion of a model in the ensemble in addition to growing:

$$l_{\text{shrink}} = \text{cross-val}(L(E[j] \leftarrow \emptyset)) \tag{7}$$

Each method then decides which action to perform: replace, grow (BG and BD only) or shrink (BD only).

3.3 Integration function

In SEGO, as it was designed for classification, a majority voting ensemble *integration function* was used. As the ensemble prediction in SEGOR is already a linear combination of its component regressors, the weights $w_{1:n}$ for ensemble size n are suitable for optimization [20]. If we express the set of weights w_i for $i = 1 : n$ as w, then by finding $\text{argmin}_w L(\sum_{i=1}^{N}(r_i(x) * w_i))$ it is possible to find weights minimizing the validation loss, with an increased risk of overfitting the validation set, however. For the hyperparameter tuning step mean-weighting is used as otherwise the individual models' equal impact on the loss-evaluation is no longer guaranteed. We explore weighting using Python's SciPy implementation of BFGS [13].

4 Experiments

The objective of the experimentation is two-fold: Firstly, we want to demonstrate the applicability of SEGO to regression and secondly highlight the performance and robustness of our improvements. Three different types of predictors are used as base-models for hyperparameter tuning tested on 16 small to medium sized datasets. As each experiment requires training a significant number of models in addition to the optimization overhead, and computational resources

were limited for this research, the scope of experimentation had to be restricted. Therefore every dataset was limited to 2999 instances. Each experiment is 5-fold cross validated and the number of Bayesian optimization iterations was set to 100 for every tested approach. The following shorthand will be used: *method - loss-function*, where s is a single model and f represents SEGOR with a fixed ensemble size of 12; g and d are the BG and BD ensemble size methods from Section 3.2, respectively. The loss functions m, h and t correspond to MSE, Huber loss and Tukey's bisquare. Lastly, the suffix ph refers to the ensemble generated post-hoc using PHEG.

All regression models were chosen from the scikit-learn library [18]. The base-models and their hyperparameters are:

- DecisionTree (DT):
  max_depth, max_features, min_samples_split, min_samples_leaf
- MultiLayerPerceptron (MLP):
  learning_rate, activation, hidden_layer_sizes (max. 2 hidden layers)
- ElasticNet (EN):
  alpha, l1_ratio, max_iter

The datasets and the number of used features and instances can be viewed in Table 1. Unless cited otherwise, they were retrieved from the UCI Machine Learning Repository [16]

Dataset	Instances	Features
CPU (cpu)	209	6
Boston Housing (bos)	506	13
White Wine (ww)	2999	11
Red Wine (rw)	1599	11
Chicago Speed (csp) [24]	2999	3
MPG (mpg)	398	6
Power Plant (pow)	2999	4
Solar Flare(sf)	1066	23
Facebook Comments (fp)	500	10
Air Quality (aq)	2999	12
Concrete Stength (cs)	1030	8
Cooling Efficiency (ce)	768	8
Heating Efficiency (he)	768	8
Math Grades (mg)	395	56
Yacht Resistance (yr)	308	6
Forest Fire Area (ffa)	517	22

Table 1: Datasets

A single predictor (s-m) optimized for 100 iterations and a post-hoc ensemble generated from its history (s-m-ph) will serve as a baseline. The performance is measured as the MSE of previously normalized values, irrespective of the loss function used. Furthermore, the diversity of the ensemble will be measured by the mutual information (MI) of the individual models' predictions. In order to

measure MI of an ensemble, the sum of all pairwise MI is taken, divided by the ensemble size squared [8]. Low values of MI represent ensembles with diverse predictions.

$$MI(Y^m; Y^n) = -\frac{1}{2} log(1 - \rho^2) \tag{8}$$

where Y^i is the prediction of model i, and ρ is the correlation between two predictors:

$$\rho = \frac{\sum_{i=1...N}(y_i^m - \mu_Y^m)(y_i^n - \mu_Y^n)}{\sqrt{\sum_{i=1...N}(y_i^m - \mu_Y^m)^2 (y_i^n - \mu_Y^n)^2}}. \tag{9}$$

4.1 Performance Results

	s-m	f-m	g-m	d-m	s-m-ph	f-h-ph	g-h-ph	d-h-ph	f-t-ph	g-t-ph	d-t-ph
DT	10.19	4.12	4.56	5.50	6.19	5.69	5.88	7.00	4.81	5.44	6.62
MLP	9.38	7.25	4.81	3.88	5.56	6.31	6.50	5.69	5.88	4.69	6.06
EN	7.00	6.00	6.81	6.00	6.81	6.06	6.00	5.06	5.38	5.25	5.44

Table 2: Mean ranks over 3 hyperparameter spaces

In order to compare the different algorithms, their performance was measured for each of the base-models separately. We compare the combination of different sizing methods and loss functions.

For methods using loss functions other than MSE, a post-hoc ensemble is used to measure performance, as the PHEG procedure can utilize the same performance-function as the final benchmark. Therefore, post-hoc results are used where appropriate to ensure methods are not disadvantaged. An intuitive and insightful way to compare different algorithms is to observe their mean ranks, which are shown in Table 2.

In order to measure performance differences we used two procedures. The first is a two-step process for comparing multiple methods simultaneously. First, a Friedman test is used to test whether there is a significant difference between all methods [7]. In principle, the test considers the mean rank, and is a non-parametric version of the well known ANOVA test. The p-values are 2.8×10^{-5}, 6.0×10^{-4} and 0.72 for DT, MLP and EN respectively. This suggests that for elastic nets, all methods perform similarly. Further investigation revealed that elastic nets performed worse in absolute terms compared to the other base types.

When the Friedman test shows significant differences, a post-hoc test is utilized to determine which methods differ from one another. The Nemenyi test can be used to compare all methods based on mean ranks [7], and includes a compensation for multiple comparisons. Our findings show that the Nemenyi test is more conservative, and is very sensitive to the methods selected for comparison. For instance, *d-h-ph* no longer outperforms *s-m* at $p < 0.05$ for DT space. In the

case of MLPs, *f-m*, *f-h-ph*, *g-h-ph*, *d-h-ph*, *f-t-ph* and *d-t-ph* are all above the critical p-value when compared to *s-m*. All performance differences are insignificant even at $p < 0.10$ for elastic nets.

Table 3 shows the results of the Nemenyi post-hoc test, comparing all proposed methods with the single optimized model *s-m*. All other comparisons values are insignificant at $p > 0.10$, and are not shown but were performed simultaneously.

	f-m	g-m	d-m	s-m-ph	f-h-ph	g-h-ph	d-h-ph	f-t-ph	g-t-ph	d-t-ph
DT	0.00	0.00	0.00	0.03	0.00	0.00	0.13	0.00	0.00	0.04
MLP	0.77	0.00	0.00	0.05	0.24	0.33	0.06	0.10	0.00	0.15

Table 3: Nemenyi test p-values for both hyperparameter spaces

The second procedure, the Wilcoxon signed-ranks test is a non-parametric paired test, which can compare the performance of two algorithms given multiple instances [7]. The methodology considers the absolute value of the performance difference, and is therefore more representative than mean ranks. Furthermore, it does not assume sampling from a normal distribution, as the paired t-test would.

In summary, all methods outperform the single optimized model, with $p < 0.05$ for both decision trees and multi-layer perceptrons. For elastic nets however, all methods show insignificant improvement at $p < 0.05$ in the pairwise comparison. Notable is the performance of *f-m* which outperforms *s-m-ph* at $p < 0.05$ for decision trees, therefore outperforming an ensemble generated using an OPAS method. Similarly *g-m* and *d-m* perform better than *f-m* for multi-layer perceptrons. This highlights the importance of non-fixed ensemble sizing for certain domains. Tables 4 and 5 show all pairwise p-values of the Wilcoxon tests, the structure of the tables are such that a value in row i and column j represents the p-value that algorithm i outperforms algorithm j.

	s-m	f-m	g-m	d-m	s-m-ph	f-h-ph	g-h-ph	d-h-ph	f-t-ph	g-t-ph	d-t-ph
s-m		1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
f-m	0.00		0.47	0.05	0.00	0.07	0.02	0.01	0.43	0.03	0.00
g-m	0.00	0.55		0.06	0.07	0.19	0.11	0.01	0.70	0.14	0.03
d-m	0.00	0.96	0.94		0.15	0.70	0.43	0.19	0.74	0.63	0.55
s-m-ph	0.00	1.00	0.94	0.86		0.72	0.70	0.23	0.83	0.57	0.74
f-h-ph	0.00	0.94	0.83	0.32	0.30		0.08	0.01	0.75	0.45	0.03
g-h-ph	0.00	0.99	0.90	0.59	0.32	0.93		0.05	0.86	0.74	0.49
d-h-ph	0.00	0.99	0.99	0.83	0.78	0.99	0.95		1.00	0.99	0.96
f-t-ph	0.00	0.59	0.32	0.28	0.19	0.26	0.15	0.00		0.47	0.01
g-t-ph	0.00	0.97	0.87	0.39	0.45	0.57	0.28	0.01	0.55		0.15
d-t-ph	0.00	1.00	0.97	0.47	0.28	0.97	0.53	0.05	0.99	0.86	

Table 4: Pairwise Wilcoxon test p-values for decision trees

	s-m	f-m	g-m	d-m	s-m-ph	f-h-ph	g-h-ph	d-h-ph	f-t-ph	g-t-ph	d-t-ph
s-m		0.99	1.00	1.00	1.00	1.00	0.99	1.00	0.99	1.00	1.00
f-m	0.01		0.97	0.99	0.55	0.65	0.28	0.65	0.47	0.87	0.74
g-m	0.00	0.03		0.70	0.19	0.01	0.06	0.14	0.10	0.33	0.26
d-m	0.00	0.01	0.32		0.07	0.08	0.03	0.02	0.03	0.17	0.10
s-m-ph	0.00	0.47	0.83	0.94		0.53	0.45	0.78	0.61	0.89	0.88
f-h-ph	0.00	0.37	0.99	0.93	0.49		0.35	0.57	0.37	0.85	0.43
g-h-ph	0.01	0.74	0.95	0.98	0.57	0.67		0.28	0.53	0.94	0.55
d-h-ph	0.00	0.37	0.87	0.99	0.23	0.45	0.74		0.55	0.75	0.68
f-t-ph	0.01	0.55	0.91	0.98	0.41	0.65	0.49	0.47		0.65	0.41
g-t-ph	0.00	0.14	0.68	0.84	0.12	0.16	0.07	0.26	0.37		0.13
d-t-ph	0.00	0.28	0.75	0.91	0.13	0.59	0.47	0.33	0.61	0.88	

Table 5: Pairwise Wilcoxon test p-values for multi-layer-perceptrons

We have excluded the weighted methods from the above comparisons, as we can observe weak performance, likely due to overfitting on the validation data or multi-collinearity among the predictors [17].

In conclusion, the proposed methods work well on the DT and MLP hyperparameter space, where the tunable parameters strongly affect the way the regressor learns. The sizing methods performed well, with *g-m* and *d-m* trading spots depending on the hyperparameter space. Robust loss functions did not improve generalization accuracy overall, and the Nemenyi test indicates they do not always perform significantly better than a single optimized regressor. We suspect this is related to the relatively small datasets. We believe given more diverse data or if the real loss function cannot be used for learning, robust loss functions would be more applicable.

4.2 Diversity

We can utilize the MI diversity measure to explain the poor performance of the algorithms on the EN hyperparameter space. Table 6 highlights the much greater mean mutual information in the ensemble of elastic nets. This corresponds with the design of elastic nets, where the impact of different parameters has a limited effect on the prediction. Therefore, our hyperparameter tuning focus does not work well for such base-models, and the MI measure can be used for detecting this situation.

	dt	nn	en
mean mut. info.	1.35	1.42	4.28

Table 6: Mean distance measures for the hyperparameter spaces

As all of the aforementioned experiments were conducted with a budget of 100 iterations, the ensemble methods do not converge. We conducted an experiment

using the 'rw' dataset for decision trees, with 300 iterations. The fixed-ensemble-size method *f-m* was tested for all sizes in the range $[4, 28]$, with sizes 25, 22 and 19 being the top performers. Our BG method *g-m* ranked 5th, with a final mean ensemble size of 27.76. Interestingly, a fixed size of 24 would have resulted in the 20th rank for *f-m*. This high sensitivity to the a-priori fixed ensemble size highlights the importance of the dynamic ensemble sizing methods. Due to computational limitations, we were unable to investigate true convergence.

5 Conclusion

We presented a method to simultaneously generate ensembles and tune the hyperparameters of its models for regression problems. Furthermore, we introduce robust loss functions and different methods of determining the size of the ensemble on-the-fly. For models with tunable hyperparameter spaces, our proposed techniques significantly outperform single regressors. The proposed sizing methods allow the algorithm to operate without a fixed a-priori ensemble size, a parameter which was shown to impact performance. The proposed robust loss functions have failed to exceed the performance of procedures using MSE, but tend to outperform single models. For models where hyperparameters only slightly affect the diversity in predictions, the suggested methods cannot significantly improve on a single tuned predictor.

Noteworthy is the finding that depending on the chosen base-learner and dataset, a different approach might be the most suitable, highlighted by differences found between decision trees and multi-layer-perceptrons. Most importantly, however, our research demonstrates the suitability of hyperparameter tuning to regression, and showcases the performance of automated meta-learning algorithms, specifically for ensemble generation with no fixed size.

Future Research

With more computational resources available, the proposed methods could be investigated on a greater breadth of larger datasets with more optimization iterations. The utilization of a diversity or complexity term during optimization has had mixed results in other applications [10], but could nonetheless be used to aid in dynamic ensemble sizing. For highly complex hyperparameter spaces, neural networks are suitable replacements for GPs [23]. Our research focused little on ensemble weighting, but other techniques such as stacking or regularized weighting should not be discarded. Furthermore, the suitability of hyperparameter tuning and more advanced techniques such as SEGO(R) have not been extensively explored for deep neural networks, an area which we hope will see increased attention in the future.

Acknowledgements

We want to thank Mediaan for supporting this research and graciously providing compute resources.

Bibliography

[1] Belagiannis, V., Rupprecht, C., Carneiro, G., Navab, N.: Robust optimization for deep regression. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2830–2838 (2015)

[2] Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. Journal of Machine Learning Research **13**(Feb), 281–305 (2012)

[3] Bergstra, J.S., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Advances in Neural Information Processing Systems, pp. 2546–2554 (2011)

[4] Brochu, E., Cora, V.M., De Freitas, N.: A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. arXiv preprint arXiv:1012.2599 (2010)

[5] Caruana, R., Munson, A., Niculescu-Mizil, A.: Getting the most out of ensemble selection. In: Data Mining, 2006. ICDM'06. Sixth International Conference on, pp. 828–833. IEEE (2006)

[6] Caruana, R., Niculescu-Mizil, A., Crew, G., Ksikes, A.: Ensemble selection from libraries of models. In: Proceedings of the twenty-first international conference on Machine learning, p. 18. ACM (2004)

[7] Demšar, J.: Statistical comparisons of classifiers over multiple data sets. Journal of Machine learning research **7**(Jan), 1–30 (2006)

[8] Dutta, H.: Measuring diversity in regression ensembles. In: IICAI, vol. 9, p. 17p (2009)

[9] Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Advances in Neural Information Processing Systems, pp. 2962–2970 (2015)

[10] Gu, S., Cheng, R., Jin, Y.: Multi-objective ensemble generation. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **5**(5), 234–245 (2015)

[11] Huber, P.J., et al.: Robust estimation of a location parameter. The Annals of Mathematical Statistics **35**(1), 73–101 (1964)

[12] Johansson, U., Löfström, T., Boström, H.: Overproduce-and-select: The grim reality. In: Computational Intelligence and Ensemble Learning (CIEL), 2013 IEEE Symposium on, pp. 52–59. IEEE (2013)

[13] Jones, E., Oliphant, T., Peterson, P., et al.: SciPy: Open source scientific tools for Python (2001–). URL `http://www.scipy.org/`

[14] Lacoste, A., Larochelle, H., Laviolette, F., Marchand, M.: Sequential model-based ensemble optimization. arXiv preprint arXiv:1402.0796 (2014)

[15] Lévesque, J.C., Gagné, C., Sabourin, R.: Bayesian hyperparameter optimization for ensemble learning. In: A. Ihler, D. Janzing (eds.) Proceedings of the Thirty-Second Conference (2016) on Uncertainty in Artificial Intelligence, pp. 437–446. AUAI Press (2016)

[16] Lichman, M.: Uci machine learning repository (2013). URL `http://archive.ics.uci.edu/ml`

[17] Mendes-Moreira, J., Soares, C., Jorge, A.M., Sousa, J.F.D.: Ensemble approaches for regression: A survey. ACM Computing Surveys (CSUR) **45**(1), 10 (2012)
[18] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
[19] Rokach, L.: Ensemble-based classifiers. Artificial Intelligence Review **33**(1-2), 1–39 (2010)
[20] Seni, G., Elder, J.F.: Ensemble methods in data mining: improving accuracy through combining predictions. Synthesis Lectures on Data Mining and Knowledge Discovery **2**(1), 1–126 (2010)
[21] Shahriari, B., Swersky, K., Wang, Z., Adams, R.P., de Freitas, N.: Taking the human out of the loop: A review of bayesian optimization. Proceedings of the IEEE **104**(1), 148–175 (2016)
[22] Snoek, J., Larochelle, H., Adams, R.P.: Practical bayesian optimization of machine learning algorithms. In: Advances in neural information processing systems, pp. 2951–2959 (2012)
[23] Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, M., Adams, R.: Scalable bayesian optimization using deep neural networks. In: International Conference on Machine Learning, pp. 2171–2180 (2015)
[24] Speed camera violations, chicago data portal. `https://data.cityofchicago.org/Transportation/Speed-Camera-Violations/hhkd-xvj4/data`