Surrogates for Hyperparameter Optimizers Benchmarkers on MNIST with Logistic Regression

Saleh Alwer Jurren de Groot

LIACS, Leiden University, The Netherlands

Abstract

Evaluating hyperparameter optimisation models is limited by the computational cost of assessing the fitness a certain configuration on a benchmark . Eggensperger et al. [2015] propose training a surrogate model to mimic the performance of said benchmark; they test 8 baseline learners. This paper reimplements gathering data and the training of the best two models as benchmark-simulating surrogates. Furthermore, we train a XGBoost-based surrogate, and find promising results when comparing optimisers' performance on the surrogates.

1 Introduction

Hyperparameter optimisation (HPO) is one of the corner-stone tasks in machine learning. A classification/regression model is only as effective as it's set parameters. Weerts et al. [2020] studied the effect of HPO; concluding their study, they state "results show that using our computed default value often results in inferior performance compared to tuning the hyperparameter." Year by year, the number of HPO methods developed increases; it is quite computationally expensive to evaluate these methods by running them on benchmarks of classification problems, sometimes needing over 340 hours [Eggensperger et al., 2015]. The development of quality HPO methods relies on cheap processes of evaluating them.

Eggensperger et al. [2015] presents an approach to solve this issue: Train cheap-to-evaluate surrogates to mimic the behaviour of real hyperparameter optimization benchmarks. They present 8 models for training a surrogate on 9 benchmarks. Their results showed that the scikit-learn implementation of Random Forest (`RF`) and Gradient Boosting (`GB`) train the surrogates that mimic the real benchmarks most closely. Since the publishing of the paper, a gradient boosting Python package, XGBoost (`XGB`) [Chen and Guestrin, 2016], which works differently than `GB`, was developed. In this paper we aim to train the best performing models found by Eggensperger et al. [2015] (`RF` and `GB`) on one of the nine benchmarks. Furthermore, our main contribution is the training of a new model, `XGB`, on the same benchmark. Finally, the main goal of this paper is to evaluate the three models based on the similarity of performance of three optimizers on the surrogates they produce and the performance on the real benchmark. The scope of this study on training surrogates is limited to the logistic regression classifier on the MNIST dataset ([Deng, 2012]) benchmark.

2 Related Work

Eggensperger et al. [2015] find that the benchmark simulator surrogate models mimic perform similar to some extent. Section 2.4 elaborates on the paper's results.

2.1 Base Learners

A base learner is a baseline optimization problem to compare hyperparameter optimizers. Eggensperger et al. [2015] propose 9 base learners. These are shown on table 2.

Table 1: Base learners proposed by Eggensperger et al. [2015]

Algorithms	Configuration Dimension	Datasets
OnlineLDA, Log. Reg., Log. Reg. 5CV	3 to 4	MNIST
HP-NNET, HP-NNET 5CV	14	mrbi and convex
HP-DBNET	36	mrbi and convex

The machine learning algorithms are applied on the MNIST [Deng, 2012]. The evaluation of a single configuration is computationally expensive. For example, as mentioned in the research paper, onlineLDA took up to 10 hours on a single core of an Intel Xeon E5-2650 v2 CPU. Logistic regression took roughly a minute. Training HP-NNET took around 12 minutes using 2 cores with OpenBlas. A single HP-DBNET configuration evaluation took around 15 minutes on a modern (2014) Geforce GTX780 GPU.

2.2 Optimizers

Optimizers seek optimal algorithm configuration. Apart from human/grid-search, there are various optimizers. The project considers the four optimizers: random search, `SPEARMINT` [Eggensperger et al., 2013], `SMAC` (Sequential Model-based Algorithm Configuration [Hutter et al., 2011], `TPE` (Tree-Parzen Estimator) [Bergstra et al., 2011]. Further details in the supplementary material A.2.

2.3 Benchmarker Simulator Surrogate Models

The models that simulate the algorithm performance on the real datasets are: Gradient Boosting (`GB`), Random Forest (`RF`), Gaussian Process `GP`, Support Vector Regression, Nu Suppor Vector Regression (`NuSVR`), k-nearest-neighbours (`KNN`), Linear Regression `Lin. Reg.`, and Ridge Regression (`Ridge Reg.`). All but `GP` (implemented with `SPEARMINT`) are implemented using `scikit-learn`.

2.4 Results

Removing seen training data (leave-one-optimizer-out Section 3.2) gave the results below:
The benchmark simulator surrogate models do not necessarily perform worse on high dimension (two digits) than low dimensions (single digit). Some models do show a significant drop in performance, namely the regression models `Ridge Reg.`, `Lin. Reg.`, `KNN` and `NuSVR`. `RF` appears positively

Table 2: Spearman's correlation coefficient of best performing surrogate for each base learner averaged over four leave-one-optimizer-out settings [Eggensperger et al., 2015].

	onlineLDA	Log. Reg.	Log. Reg. 5CV	HP-NNET conv.	HP-NNET 5CV conv.	HP-NNET mrbi	HP-NNET 5CV mrbi	HP-DBNET conv.	HP-DBNET mrbi
Best Performing Surrogate	0.96	0.86	0.93	0.87	0.86	0.92	0.91	0.86	0.87

affected by 5-cross-fold-validation, leading to a CC increase of 0.09 in the logistic regression base learner. Typically, `RF` and `GB` perform best. In this project, we use these two surrogate models. The optimization for `TPE` and `SMAC` was sped up by approximately 20 000 times. Even SMAC, with MCMC steps, achieved a 330-fold speedup.

2.5 XGBoost

`XGB` is a state-of-the-art gradient boosting algorithm. `XGB` is a more regularized form of the standard gradient boosting [Chen, 2015]. Thus, this algorithm may be a produce another suitable surrogate model which is investigated in this project.

3 Methods

As a base learner, we reproduced the logistic regression base learner because of the computational feasibility. We select the hyperparamaters listed in Table 3 as the ones to optimize. These are the same parameters optimized by Eggensperger et al. [2015]. The data used is the MNIST dataset [reference] containing 70,000 images of handwritten digits presented as a classification task.

Table 3: Logistic regression hyperparameters.

Hyperparameter	Range
Learning Rate	0-1
L2 regularization ratio	0-1
N_epochs	5-2000

We use the sklearn SGD classifier model Pedregosa et al. [2011] as an implementation of logistic regression. As opposed to the orignal paper, we do not include the batch size as hyperparameter to reduce computational cost.

3.1 Gathering Training Data

We gather performance data for the surrogates by the optimization of the logistic regression algorithm using the same three optimizers used by Eggensperger et al; those are `SPEARMINT`, `SMAC`, and `TPE`.

Using each optimizer, we conduct 10 runs with each run using 100 evaluations. The configuration used for an algorithm evaluation gives a loss on the selected dataset, and we store the configuration-loss pair of every evaluation. We also conduct 10 runs of random search, using 100 evaluations each. This is done so that the data we gather is not too optimistic. We, therefore, have 4000 data points with algorithm configuration as features and the loss as target. Table 5 in A.3 shows an example of 3 data points.

3.2 Surrogates

Leave-one-optimizer-out (leave-ooo) is when we leave the data gathered by one optimizer during the training process of a surrogate. Otherwise, the surrogate will over-perform on the optimizer as it was trained using data generated by said optimizers. This means for every model, we train two surrogates: one without the data gathered by TPE and one without the data gathered by SMAC. We use the left out data for measuring performance. This is seen in Section 4.1

For each surrogate model, we run random search with 100 iterations to optimize the parameters seen in Table 6 in A.4. These 100 samples are trained on 50% of the leave-ooo data and tested on the rest. The selected parameter configuration is then used on all of the leave-ooo to train the surrogate model.

4 Experiments and Results

In Section 4.1, we evaluate the surrogates based on the predictions they make on the leave-ooo data using the metrics described in the supplementary material A.5. In Section 4.2 we conduct further analysis by comparing three optimizers' performances.

4.1 Predicted Against True Performance

The surrogates that were trained without TPE data are used to predict the loss of the configurations in the TPE data. The same process is done for the SMAC data. Table 4.1 shows the average RMSE and CC achieved by each surrogate on both optimizers' data.

Table 4: Average CC and RMSE of surrogates on leave-TPE out and leave-SMAC out data.

Model	RMSE	CC
`RF`	0.02273	0.5103
`XGB`	0.02439	0.4608
`GB`	0.02274	0.5093

Figure 1 shows, for each leave-TPE-out surrogate, it's loss predictions vs the actual loss achieved by logistic regression on the configuration-loss data gathered using TPE. Leave-TPE-out surrogate refers to the model trained without TPE data. A marker on the main diagonal represents a perfect prediction; the surrogate with it's points closer to the dotted diagonal is more accurate in predicting the loss. The best third predictions are green, the worst third are red and the rest are grey. The leave-SMAC-out plots are similar to these; they are seen in Figure 3 in the Supplementary Material A.6

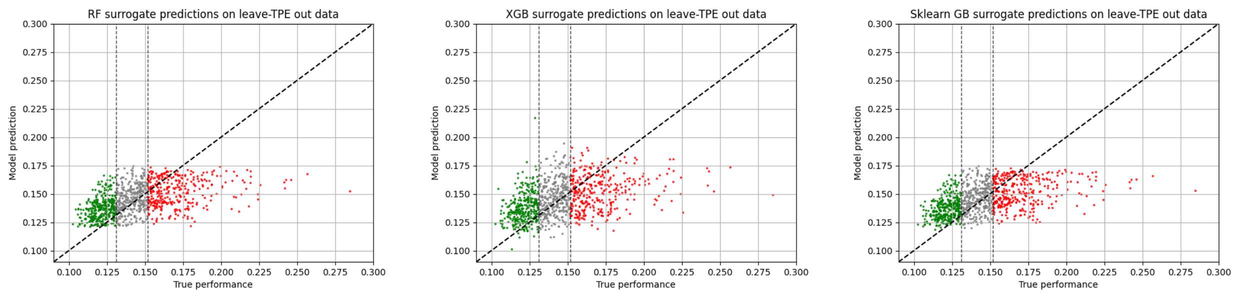

Figure 1: Each surrogate's predictions vs true loss achieved for each surrogate on TPE data.

Table 4.1 and Figure 1 both show little-to-no difference between the accuracy of the surrogates. In Figure 1, there is no significant difference between the plots for each surrogate. The RMSE for all surrogates is similar. The CC for the XGB surrogate is slightly worse than that of GB and `RF`. All three of the models perform better when predicting the good performances. The RMSE of surrogates tested in this paper cannot be compared to those in Eggensperger et al. [2015]. This is because the range of the value of loss in this paper is relatively small. This scales down the RMSE of the surrogates in this project.

4.2 Optimizers' Performance on Surrogates

Further surrogate analysis is done by looking at the performance against number of evaluations. The objective, as stated by Eggensperger et al. [2015] is for the optimizer to "perform similarly on the surrogate benchmark as on the real benchmark". Figure 2 below shows the performance of `SMAC`, `TPE` and random search. Each optimizer line is the mean of the 10 runs. The shaded area is the range within one standard deviation.
The plots suggest that `XGB` follows the real optimizer better on a high level. Particularly `GB` has an exaggerated 'drop' in best loss for the `SMAC` optimizer, and perhaps also for both other optimizers. `RF` and `GB` show convergence in performance (some optimizers to a greater extent than others). The standard deviation becomes small. However, optimization on the real data does not show this convergence. This is a limitation to both of these surrogates. `XGB` searches for longer and reaches closer to the real minimum. All three surrogates flatten out, whereas optimization on the real data still slants down. This is fair because the surrogates consider the best loss as extremum.

5 Discussion

5.1 Failed Reproduction

While the RMSE scores are actually better than those by Eggensperger et al. [2015] (due to the small range), the CC scores are almost only half. The performance of the benchmark simulator

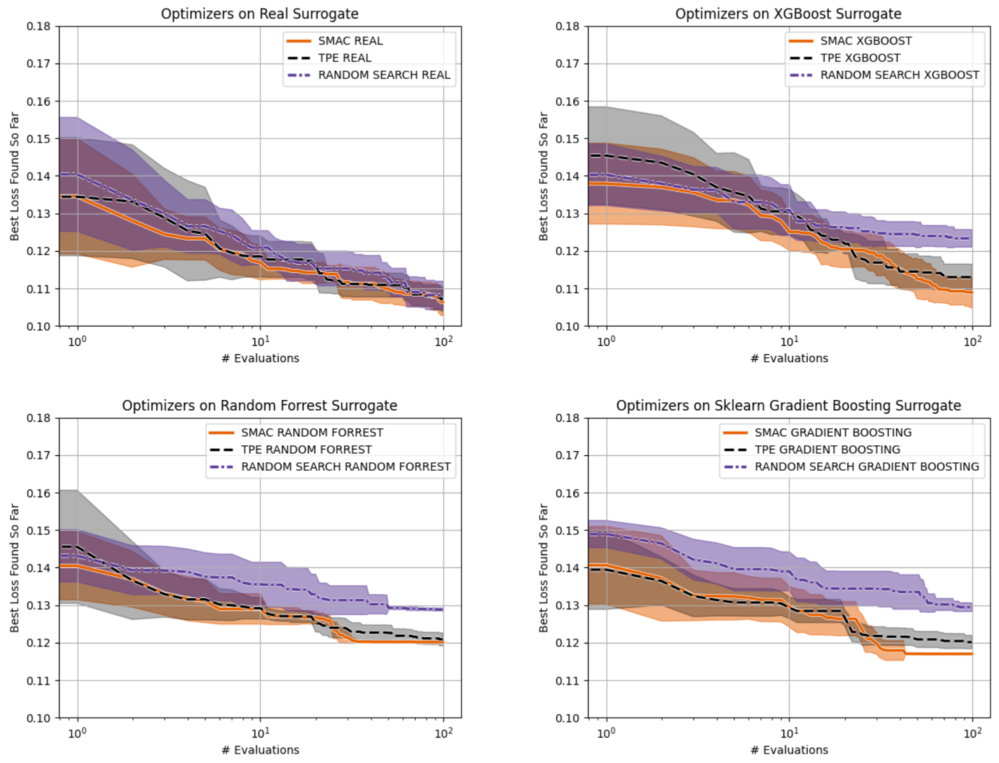

Figure 2: Performance of optimizers on the real, random forest surrogate and `XGB` surrogate.

surrogate models is thus unsatisfactory. The cause for this large difference in performance seems to be because of the `SGDClassifier` implementation, whereas Eggensperger et al. [2015] used `Lasagne` to make a neural network mimicing logistic regression. `SGDClassifier` was implemented because the algorithm allows for the same number of hyperparameters as implemented in the original paper, such as the learning rate. This classifier, however, in performs much worse than standard logistic regression. A default scikit-learn logistic regression scores 91.3% accuracy. Within the constraints of the project, a repeat of the logistic regression optimization on the original data was not deemed feasible due to the large computational cost. Further practical difficulties include the original paper's unfinished repository and conflicting optimization and regression model packages.

5.2 Surrogate Model Similarity to Real Data

Our surrogate models do not exhibit as close a resemblance to the real data as Eggensperger et al. [2015]. The plots in Section 4 do show a reasonable resemblance. Furthermore, we see a correlation in Figure 2. The performance of points predicted by `RF` and `GB` are approximately contained in the 0.125 to 0.175 range, whereas `XGB` varies from 0.100 to almost 0.225. The true performance actually varies from around 0.100 to 0.285. `XGB` appears more versatile but in this setting is outperformed by the more exploitative `RF` and `GB`. However, in the optimization of logistic regression, the true performance varies more. Thus, `XGB` may perform better with other data.

6 Conclusions and Further Research

The research paper's results were not reproduced due to the difference in error measurement and logistic regression implementation; meaning we cannot compare results between studies. From the results of this study, we can conclude training a surrogate using an `XGB` model is promising. Although it did not perform well in terms of the RMSE and CC, `SGB` and `RF` had similar results. In light of Section 4.2, the `XGB` surrogate model is promising due to its good performance in predicting extrema. We expect that in a different implementation, perhaps using a benchmark with a space of hyperparameters that give a wider range of performances, `XGB` surrogate model will perform on par with or better than `RF` and `GB`. Section 4.1 showed that all the surrogates predicted the better performances more accurately. Training the surrogates with more randomly generated configurations could solve this issue. Furthermore, this project is limited by only considering one benchmark. Testing `XGB` with more benchmarks is required in order to make a conclusive analysis.

References

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

Tianqi Chen. Tianqi chen answer to "what is the difference between the r gbm (gradient boosting machine) and xgboost (extreme gradient boosting)?". *Quora*, Sep 2015.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342322. doi: 10.1145/2939672.2939785. URL `https://doi.org/10.1145/2939672.2939785`.

Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

Katharina Eggensperger, Matthias Feurer, Frank Hutter, James Bergstra, Jasper Snoek, Holger Hoos, Kevin Leyton-Brown, et al. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In *NIPS workshop on Bayesian Optimization in Theory and Practice*, volume 10, 2013.

Katharina Eggensperger, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. Efficient benchmarking of hyperparameter optimizers via surrogates. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer, 2011.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Hilde Weerts, Andreas Mueller, and Joaquin Vanschoren. Importance of tuning hyperparameters of machine learning algorithms. 07 2020.

A Supplementary Material

A.1 Code

GitHub Repository Link : `https://github.com/j-degroot/AML_project`

A.2 Further details on optimizers

`SPEARMINT` models the performance-hyperparameter space as Gaussian process which uses a Markov Chain Monte Carlo method to marginalize over the hyperparameters. This optimizer is implementable in Python 2.7. (hard to loop, conflicting packages, logistic regression takes long on old python and does not converge quickly for given a maximum number iterations).`SMAC` uses random forest and generally performs well in non-smooth and high-dimensional problems, as opposed to `SPEARMINT`. `TPE` is relatively better on high-dimensional problems and typically outperformed by `SMAC` [Eggensperger et al., 2013].

A.3 Example of gathered data

Table 5: The top three data rows for random search.

Learning rate	L2 regularization ratio	n epochs	loss
0.239	0.491	1093	0.184
0.169	0.002	872	0.168
0.502	0.499	1977	0.203

A.4 Surrogates parameter space

Table 6: Random Search Space for Training each Surrogate.

Model	Parameters	Space
RF	`n_estimators`	Integer, [20,200]
	`min_samples_split`	Float, [0,1]
	`max_features`	Float, [0,1]
`GB`	`max_depth`	Integer, [2,25]
	`max_features`	Float, [0,1]
	`min_samples_leaf`	Float, [0,1]
`XGB`	max_depth	Integer, [3,18]
	`gamma`	Float, [1,9]
	`learning_rate`	Float, [0.005,0.5]
	`min_child_weight`	Integer, [1,10]

A.5 Evaluation Metrics

Root Mean Square Error (RMSE) is a metric for measuring the error made by $\widehat{Y}$ on Y where $\widehat{Y}$ is a set of predictions and Y is the set of targets. It is calculated using Equation 2.

$$RMSE = \sqrt{\sum_{i=1}^{|Y|} \frac{(\widehat{Y}_i - Y_i)^2}{|Y|}} \tag{1}$$

Spearman's Correlation Coefficient (CC) is a metric for measuring the similarity between two sets $\widehat{Y}$ and Y. It is calculated using Equation 2 where both sets, Y and $\widehat{Y}$, are ranked in descending order and d_i is the difference in rank for the ith prediction. The higher the CC value, the higher the similarity between the sets.

$$CC = 1 - \frac{6\sum_{i=1}^{|Y|} d_i^2}{|Y|^3 - |Y|} \tag{2}$$

A.6 Leave-SMAC-out

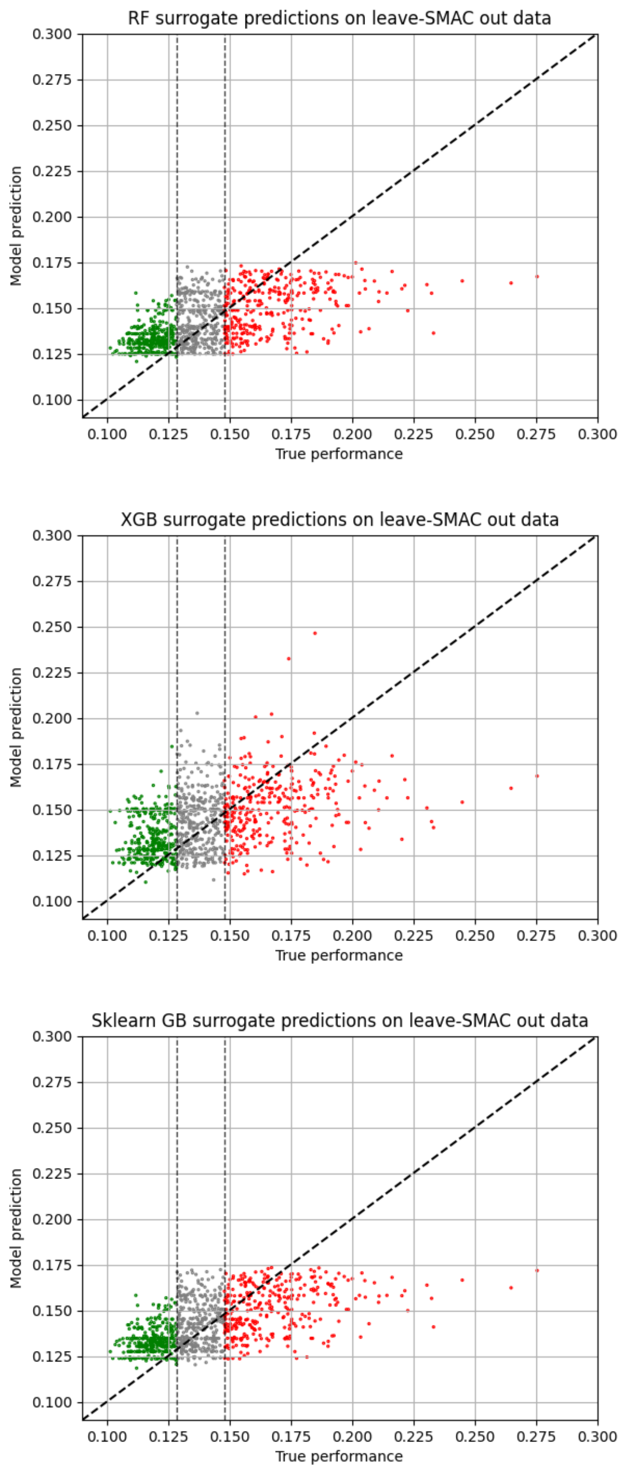

Figure 3: Each surrogate's predictions vs true loss achieved for each surrogate on SMAC data.