Faculty of Mathematics and Information Science

WARSAW UNIVERSITY OF TECHNOLOGY

Tunability: case study

AutoML 2023Z

Dawid Płudowski, Anotni Zajko

November 21, 2023

1 Methodology

In this section, we introduce methodology of experiments, explain and introduce notation used in this paper and list models and datasets used in experiments.

In our experiments, we used only binary classification tasks from OpenML [Vanschoren et al., 2013]. Each of tasks has from 1000 to 4000 records. Each of dataset is preprocessed by pipeline that filling missing values with mean (numeric columns) or most frequent value (categorical columns). Next, numeric data were standardized using z-score. Categorical data was encoded using one-hot method. All data used in experiments is listed in Table 1.

We evaluated three models: logistic regression, SVM and gradient boosting. Implementation of the models was taken from package scikit-learn [Pedregosa et al., 2011]. Hyperparameters selection and grid is presented in Table 2. Compared to [Probst et al., 2019], our grid is significantly smaller due to calculation time. Moreover, for further performance improvements, we preset some hyperparameters as shown in Table 3.

During our experiments we estimated three tunability indicators: best (default) hyperparameters θ^*, tunability of the algorithm on the j-th dataset $d^{(j)}$ and tunability of i-th hyperparameter on j-th dataset $d_i^{(j)}$. Best hyperparameters were computed as θ that gave best mean accuracy across all datasets. Rest of the indicators were computed as in [Probst et al., 2019].

2 Experiments

2.1 Algorithm tunability

2.1.1 Random search

In the first experiment, we evaluated the ability to improve model performance using the Random Search hyperparameters optimization algorithm. We evaluated tasks on baseline models (i.e. with hyperparameters set as θ^*). We used default hyperparameters except for those specified in Table 3. Next, we tune hyperparameters specified in Table 2 using the aforementioned method. As an evaluation metric, we used accuracy.

In Figure 1, we show a comparison of accuracy gains between considered ML algorithms. For cases when the searching did not found better results than defaults, we set gain to 0. In this plot, we can see that the tunability of the algorithm is highly dependent on the dataset. There is a huge difference between *steel-plates-fault* and *credit-g*, and what is more important is that this difference is consistent among all algorithms.

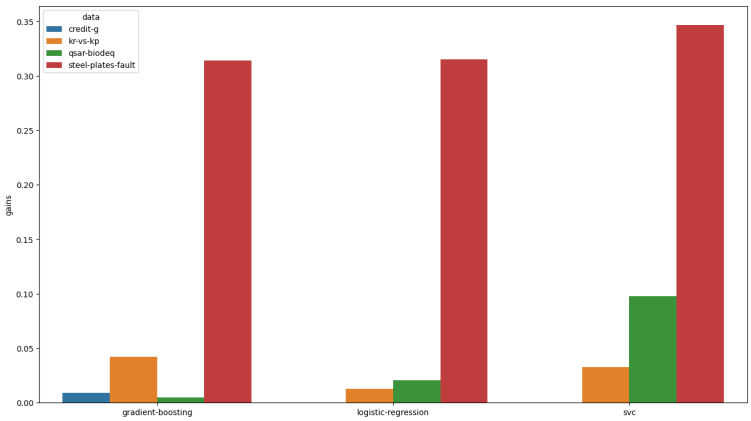

Figure 1: Comparison of the accuracy gained obtained using random search hyperparameter optimization algorithm.

2.1.2 Bayesian optimization

In the second experiment, we did analogous runs, but instead of random search, we used Bayesian optimization implemented in the scikit-opt package. As a baseline, we used the same models as in subsection 2.1.1.

In Figure 2, we show how tunable models are by Bayesian optimization. Results were similar to those in Figure 1, so also we conclude that tunability is highly dependent on data.

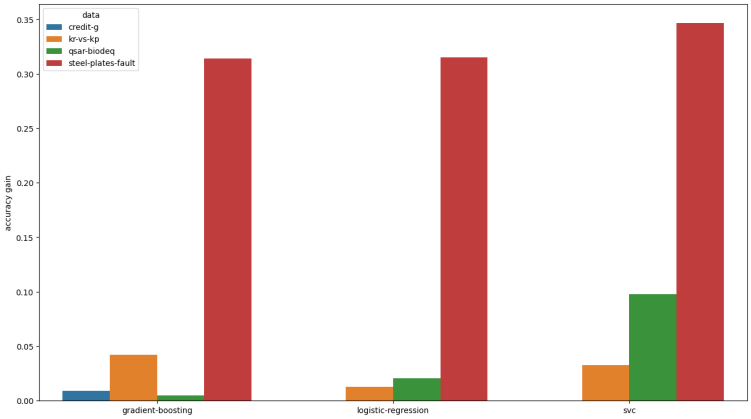

Figure 2: Comparison of accuracy gain obtained using a Bayesian optimization algorithm.

2.1.3 Convergence

In Figure 3, we show accuracy over time for all runs and, for both HO methods. We can see that all runs converged to relatively good and stable results which proves the reliability of above reasoning. Moreover, we can conclude that 30 iterations are enough for random search and 20 for bayesian optimization, to get reasonable improvement in performance (if achievable for a given model and dataset). We do not see any

significant differences in the results of the experiments that depend on the search algorithm. The reason of this is the high number of iterations that both algorithms have to search for best hyperparameters (100 iterations). Plots on Figure 3 show that bias sampling could be observed if the number of the iterations was set to less than around 30.

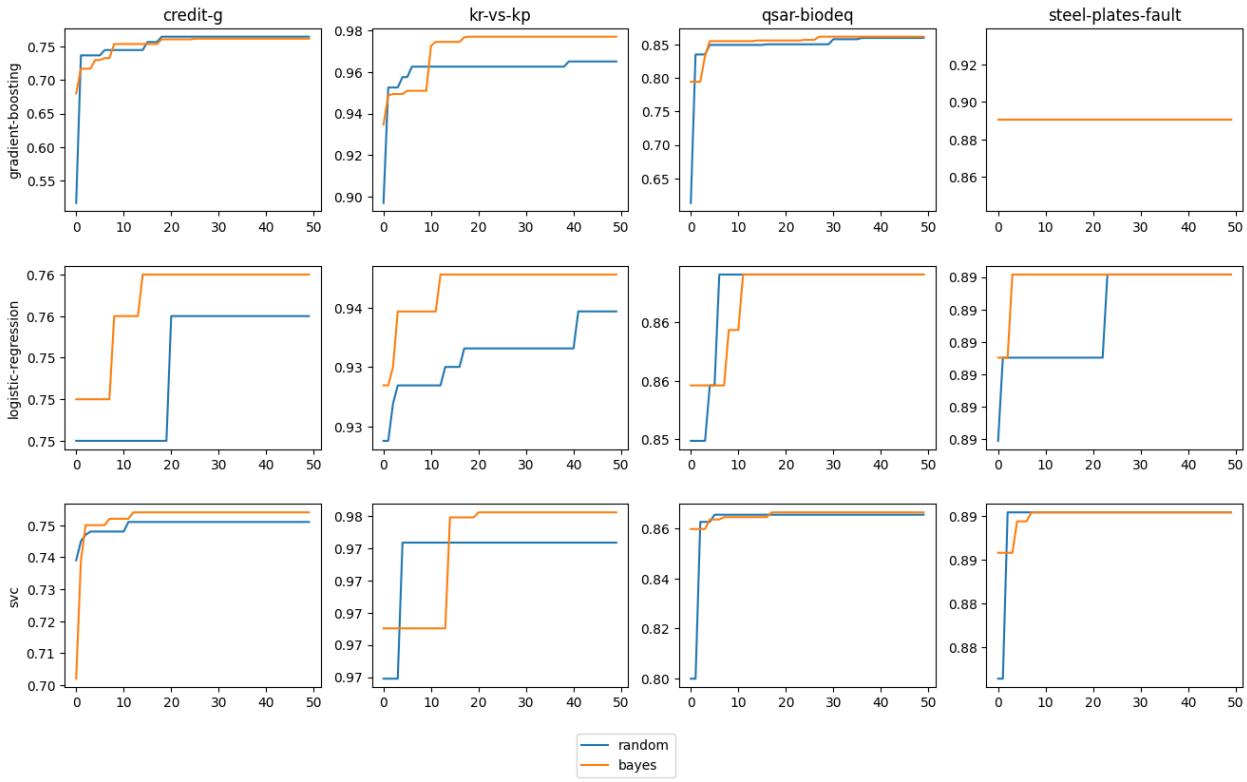

Figure 3: Accuracy over time compared for all models and datasets for all algorithms.

3 Single parameter tunability

In Figure 4, we show the tunability of a single hyperparameter using random search and in Figure 5, we show tunability for Bayesian optimization.

On these figures, we can also see that the tunability of the algorithm is much more dependent on the dataset than on the optimization algorithm. We can see that on majority of cases on *steel-plates-fault*, all hyperparameters were highly tunable, but on *credit-g*, optimization of any hyperparameter didn't bring any significant gain.

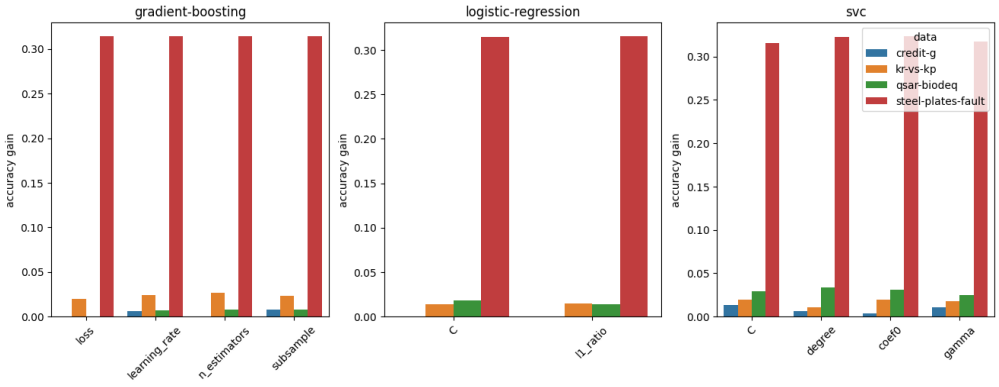

Figure 4: Comparison of accuracy gain obtained using bayesian optimization algorithm for single hyperparameters.

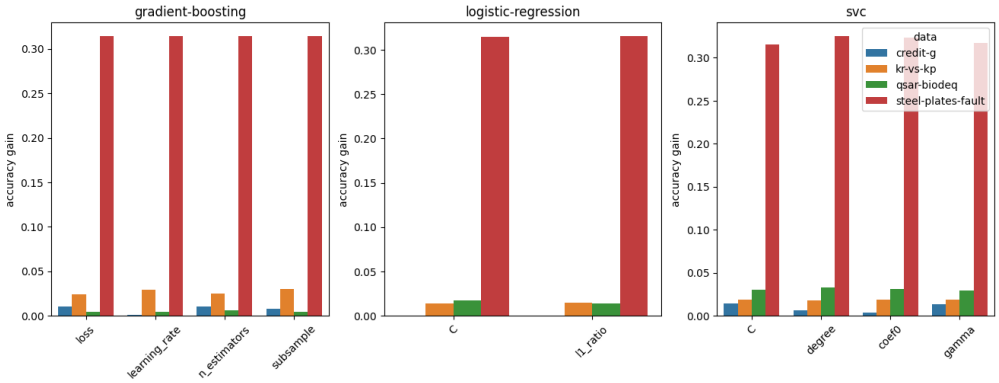

Figure 5: Comparison of accuracy gain obtained using bayesian optimization algorithm for single hyperparameters.

References

[Pedregosa et al., 2011] 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., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Probst et al., 2019] Probst, P., Boulesteix, A.-L., and Bischl, B. (2019). Tunability: Importance of Hyperparameters of Machine Learning Algorithms.

[Vanschoren et al., 2013] Vanschoren, J., van Rijn, J. N., Bischl, B., and Torgo, L. (2013). Openml: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60.

A Appendix

OpenML id	Name	# Rows	# Features
31	credit-g	1000	20
1504	kr-vs-kp	3196	36
3	qsar-biodeg	1055	41
1494	steel-plates-fault	1942	27

Table 1: Datsets used in experiments.

Model	Hyperparameter	Type	Range
Logistic Regression	C	float	[0,1]
	l1_ratio	float	[0,1]
SVM	C	float	[0,1]
	degree	float	[0,1]
	coef0	float	[0,1]
Gradient Boosting	loss	category	[deviance, exponential]
	learning_rate	float	[0,1]
	n_estimators	int	[10, 200]
	subsample	float	[0,1]

Table 2: Hyperparameters tested in experiments.

Model	Hyperparameter	Value
Logistic Regression	max_iter	10000
	solver	saga
	penalty	elasticnet
SVM	max_iter	10000
	kernel	poly
Gradient Boosting	n_iters_no_change	1000

Table 3: Hyperparameters preset during experiments.