Hypercube: a DOE-informed Hyperparameter Optimization Machine

Kaiwen Wang
M.S. Statistics
kwang460@gatech.edu

Xiaochen Yan
M.S. Statistics
xyan330@gatech.edu

Abstract

Optimal performance of novel machine learning models often relies on meticulous tuning and optimization of hyperparameters. Conventional approaches such as grid search or random search, in conjunction with cross-validation, are computationally expensive and insufficient for effect/sensitivity inference. We present a framework for hyperparameter optimization (HPO) using experiment design and analysis methods, such as Latin hypercube, response surface, and ANOVA. In preliminary results, the Python implementation (package: *hypercube*) yielded competitive results against state-of-the-art HPO packages (*optuna* and *hyperopt*) in both accuracy and budget.

I. Introduction

It is critical to first distinguish between a parameter and a hyperparameter in machine learning (ML). The former is learned directly from data. An example is coefficients for linear regression. In contrast, hyperparameters govern the overall behavior of the ML model. They cannot be learned by the model and must be set manually. An example is the penalty coefficient in Lasso regression. In subsequent sections, we may refer to hyperparameters as parameters for efficiency.

To evaluate hyperparameters' quality, a common practice is to use a cross-validation (CV) strategy: partitioning the training set into a training subset and a validation subset. The model is fitted on the training subset and evaluated on the validation subset. Such repetition of partitioning-training-validating can be carried out *k* times and with a different validation subset each time. Performance is then averaged across folds. Such a strategy is called a K-fold CV.

A high-performance ML package, *sklearn* offers grid search (*GridSearchCV*) and randomized search (*RandomizedSearchCV*) suitable for both continuous and discrete hyperparameter tuning. The former performs an exhaustive search over all combinations, given parameter value options. The latter picks random combinations, given user-defined parameter ranges/distributions. Parameter combination with the highest CV score is selected as the best combination.

These traditional approaches exhibit limited efficacy in modern ML, where the search space is often extremely complex. For grid search, irrelevant parameters can exponentially increase the number of iterations. Random search is less susceptible to the curse of dimensionality but nonetheless yields sub-optimal solutions. Regardless, both approaches are expensive and lack interpretability; inference on parameter significance and effect direction can hardly be made.

As the objective function (model performance) or its gradient cannot be explicitly evaluated, recent research frames the tuning problem as a blackbox optimization (BO) problem. Grid and random search can be considered model-free BO methods. Other applicable BO approaches use variants of Bayesian optimization–a sequential design and analysis strategy for global optimization. A surrogate model, usually Gaussian Process (GP), is used to characterize the behavior of the blackbox function. Popular Bayesian methods such as tree-structured Parzen estimator (TPE) are widely adopted by state-of-the-art HPO packages such as *hyperopt* and *optuna*.

II. Methods

A. Response Surface Method

The response surface method (RSM) is a sequential design and analysis strategy for continuous space search. It consists of an iterative process of obtaining samples, modeling the response surface using surrogates, and sampling again. A popular sub-method of RSM is the steepest ascent method.

It assumes a first-order or second-order local response and uses a gradient-based approach for greedy sampling. Linear effects are estimated in broad searches, and quadratic effects are estimated in more granular searches.

The steepest ascent method has limitations in large search spaces with blackbox objectives. The foremost relates to its convexity assumption, which is rarely satisfied in HPO. This also implies that the performance is at the expense of initializing samples. In addition, the steepest ascent itself is governed by a hyperparameter–step size, which indicates the grid layout of search space. There exist advanced gradient methods such as stochastic gradient descent and adaptive gradient, but they cannot be easily implemented in the context of experimental design.

B. Latin Hypercube Design (maximin criterion)

Another applicable design method was the combination of Latin Hypercube Design. Latin Hypercube Design efficiently allocates experimental runs by ensuring that they are evenly spread across the experimental region – the space defined by all possible combinations of factor levels. The experimental region is divided into smaller uniform slices, and each of them receives at least one sample point, ensuring that the entire range of each factor is explored.

However, a traditional LHD can be too random to effectively fill the space in two or more dimensions. Thus, we incorporate the Maximin criterion, which focuses on maximizing the minimum distance between any two points in a design, thereby ensuring a more even and space-filling distribution of points. This approach helps in overcoming the randomness of a traditional LHD. This hybrid approach offers an advantage over Orthogonal Array (OA) LHDs, as it allows for more flexibility in choosing the sample size (n), provided n is at least as large as the number of factors (d). This flexibility makes maximin LHDs a more versatile and efficient choice for experimental designs, especially in scenarios involving complex multivariate spaces.

C. Methods for Categorical Factors

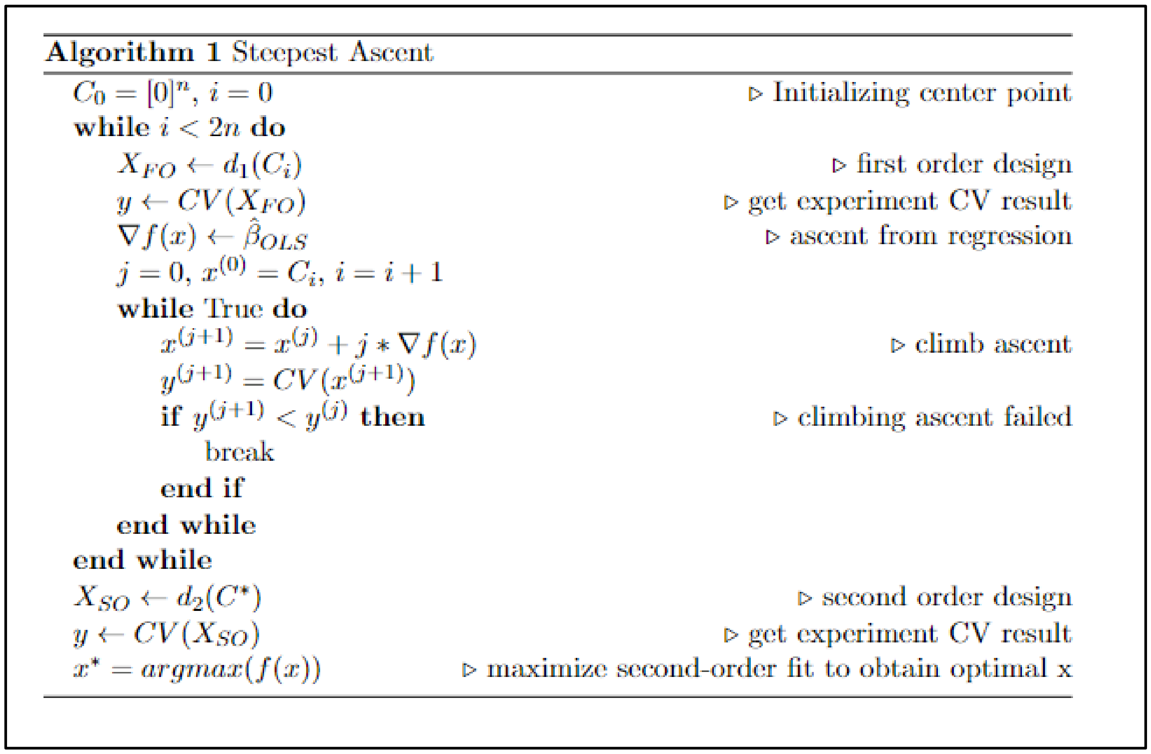
Categorical factors impose unique constraints on many established optimization techniques. Experiment design on categorical factors often employs full or fractional factorial design. Low-budget techniques including orthogonal main effect and Plackett-Burman designs can be restrictive in the number of factors or number of factor levels. For HPO, decision space can be arbitrarily nonregular, making it hard to generalize to cost-effective methods. For analysis, traditional approaches such as regression and ANOVA remain the most reliable methods. Under ordinary least squares (OLS) regression, coefficients, p-values for coefficients, and R^2 can be obtained in closed form. Respectively, they represent the direction of effect, significance of effect, and quality of fit. An extension of regression, ANOVA investigates the variance composition of the response. The greater relative sum of squares (or mean squares) indicates a greater effect exerted by the factor; this can be evaluated through an F-test.

In the context of hyperparameter tuning, purely discrete parameters (e.g. loss criterion in Random Forest: “entropy” or “gini”) are rare. For integer-valued parameters (e.g. number of estimators in Random Forest), it is generally recommended to conduct a continuous space search with proper rounding instead of a discrete search. Discretizing inevitably leads to an exhaustive grid search, an expensive practice that is challenging to optimize.

III. Implementation

A. Response Surface Method

Hypercube offers an implementation of RSM–the steepest ascent in the *Surf* class. Both continuous (float) and integer-valued parameters are supported. The algorithm is as follows:



```
**Algorithm 1** Steepest Ascent
$C_0 = [0]^n$, $i = 0$                                   ▷ Initializing center point
while $i < 2n$ do
    $X_{FO} \leftarrow d_1(C_i)$                          ▷ first order design
    $y \leftarrow CV(X_{FO})$                             ▷ get experiment CV result
    $\nabla f(x) \leftarrow \hat{\beta}_{OLS}$           ▷ ascent from regression
    $j = 0$, $x^{(0)} = C_i$, $i = i + 1$
    while True do
        $x^{(j+1)} = x^{(j)} + j * \nabla f(x)$           ▷ climb ascent
        $y^{(j+1)} = CV(x^{(j+1)})$
        if $y^{(j+1)} < y^{(j)}$ then                     ▷ climbing ascent failed
            break
        end if
    end while
end while
$X_{SO} \leftarrow d_2(C^*)$                              ▷ second order design
$y \leftarrow CV(X_{SO})$                                 ▷ get experiment CV result
$x^* = argmax(f(x))$     ▷ maximize second-order fit to obtain optimal x
```

Figure 1: Surf Algorithm

Note that for higher dimensions (n>3), a subset of the Plackett-Burman design is used for the corner points in the second-order design. This does not concern our experiment below, which is conducted in the two-dimension space.

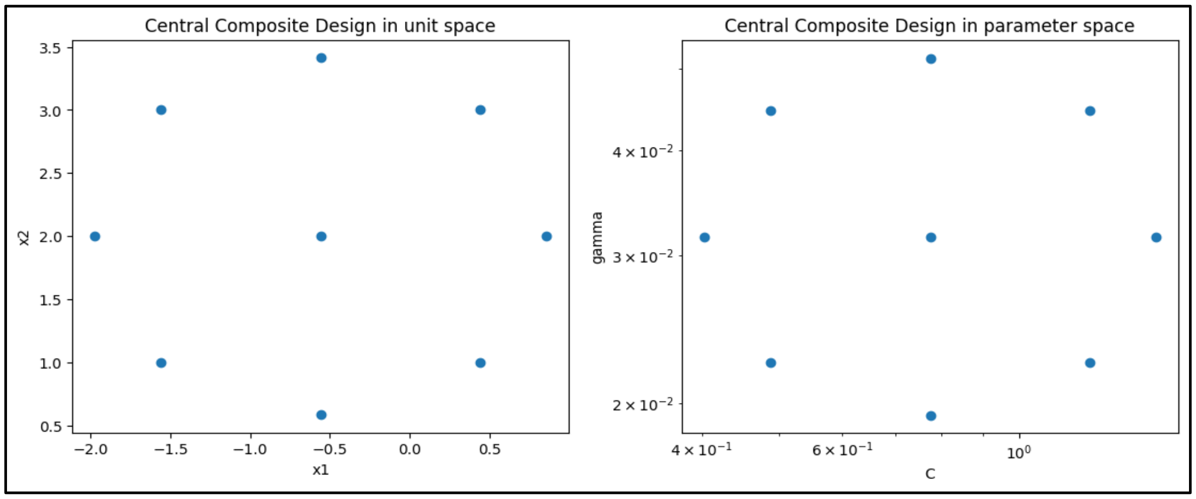


Figure 2: Central Composite Design for SVC Tuning; Conversion from Unit Space to Parameter Space

B. Latin Hypercube Design (maximin criterion)

The *LHSTuner* (or *LHS*) is method class written based on Latin hypercube sampling with maximin criterion. The sampling step is executed via another package: *pyDOE*. The class is initialized with configurations including the model, hyperparameter range, evaluation metric, CV settings and sample size. The class generates optimized sampling points, scales them as per hyperparameter specifications, and then conducts the model training and validation (per CV setting). Experimental results are then analyzed using a user-specified method (linear model, ANOVA, or GP). The analysis locates and outputs a set of optimal hyperparameters.

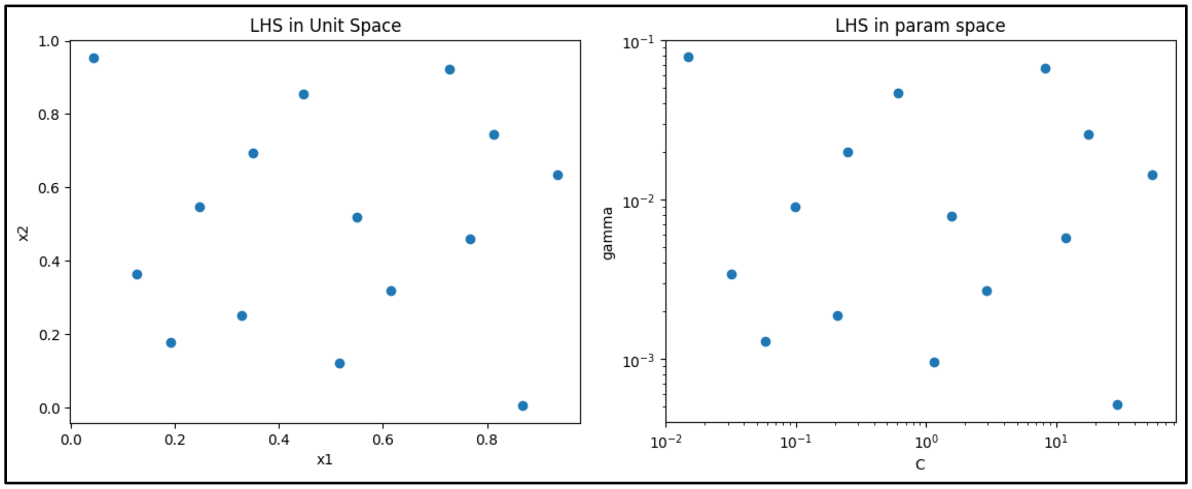


Figure 3: Latin Hypercube Sampling for SVC Tuning; Conversion from Unit Space to Parameter Space

C. One-Factor-At-a-Time (OFAT)

Hypercube also offers a full factorial design and analysis in the *OFAT* class. Note that the *OFAT* method in DoE is synonymous with grid search in ML. While this implementation does provide additional analysis methods (regression and ANOVA), there is no further value-added compared to its counterpart by *sklearn*. For convenience, we will omit further evaluations of this method.

IV. Analysis

In the following analysis, we applied the design methods to two classification models — random forest (RF) and support vector classifier (SVC). Using *sklearn*'s *make_classification* function, we generated a synthetic dataset with 300 samples. Each sample has 10 features and is classified into two classes suitable for a binary classification problem.

There were 2 hyperparameters to be optimized for the RF model:

- Minimum sample leaf: integer, range from 2 to 10
- Number of estimators: integer, range from 30 to 100

Before and after sample generation, range transformations to and from unit space were carried out, with proper rounding to adapt to the parameter's discrete data type.

There were 2 hyperparameters to be optimized for the SVC model:

- C: log10, range from 0.01 to 100
- Gamma: log10, range from 0.0005 to 0.5

We conducted 5-fold and 4-fold cross-validation for the RF and SVC models respectively. For LHS, 10 samples were drawn from the sample space for each run.

Table 1 and 2 summarize performance comparisons of *LHS* and *Surf* after one experiment study. Both methods obtained results in close proximity to the global optimum obtained by grid search. It is also observable that *Surf* method is more computationally expensive, compared to *LHS*.

Comparison of Design Method Efficiency (RF)			
	Total Runs	Run Time (s)	CV Score
Grid Search	3195	147.4	0.8367*
LHS	50	2.3	0.8270
Surf	150	9.9	0.8260

Table 1: Comparison for Random Forest Experiment

Comparison of Design Method Efficiency (SVC)			
	Total Runs	Run Time (s)	CV Score
Grid Search	25600	179.4	0.8550*
LHS	40	0.2	0.8352
Surf	88	0.3	0.7995

Table 2: Comparison for SVC Experiment

Figure 4 serves to compare different design methods using multiple experiment studies. In each study, ten points were sampled within the experimental space and an optimum is obtained after analysis. Each study is reproduced five times for a method. This is then repeated for each method among *random search, hyperopt, optuna, and LHS*. Since the *Surf* method is deterministic, given that the initial point stays the same for each run, only 1 optimal point of this method is shown in the graph (also note that *Surf* is unconstrainted by budget). The color-coded background is the analysis outcome with an exhaustive grid search over the entire experiment space. One can interpret this as the ground truth response surface.

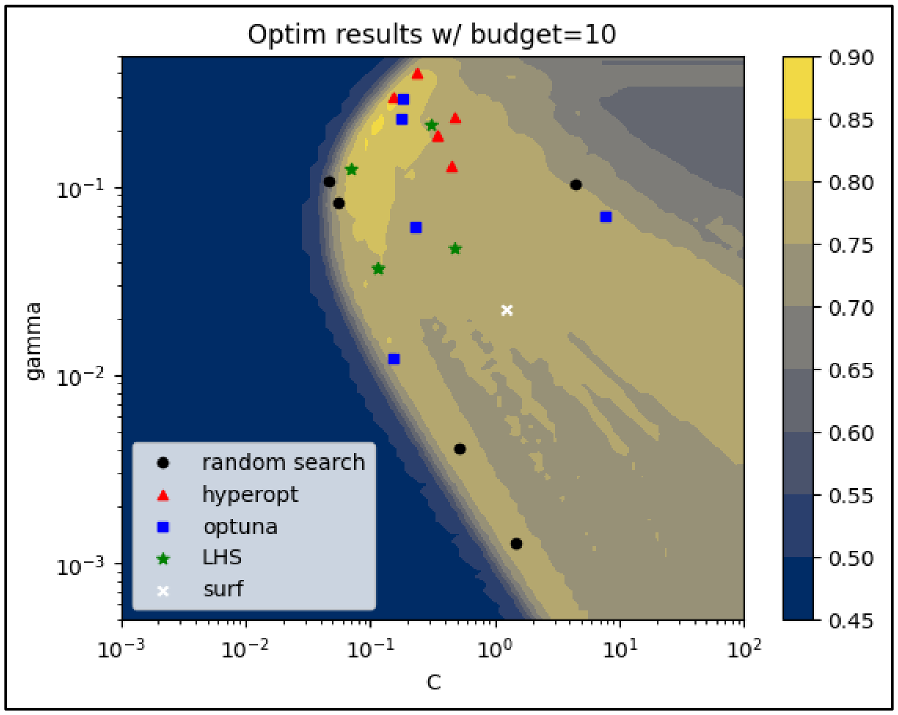


Figure 4: Replicated Method Comparison for SVC Experiment

The global optimum is located at around C=0.1 and gamma=0.2. In general, all 5 design methods produced high-quality results, as all optimization results yield >0.75 CV score. In particular, the sets of optimal points of *LHD* and *hyperopt* concentrate around the global optimum. The results from *optuna* and *random search* exhibit higher estimation variance. Note that the *Surf* algorithm did not make much movement beyond the center.

LHS's implementation of the GP is different from Bayesian optimization. In its current version, the algorithm is only interested in taking the global argmax of GP and outputting it as the optimal hyperparameters. We further extend this to a response surface approximation.

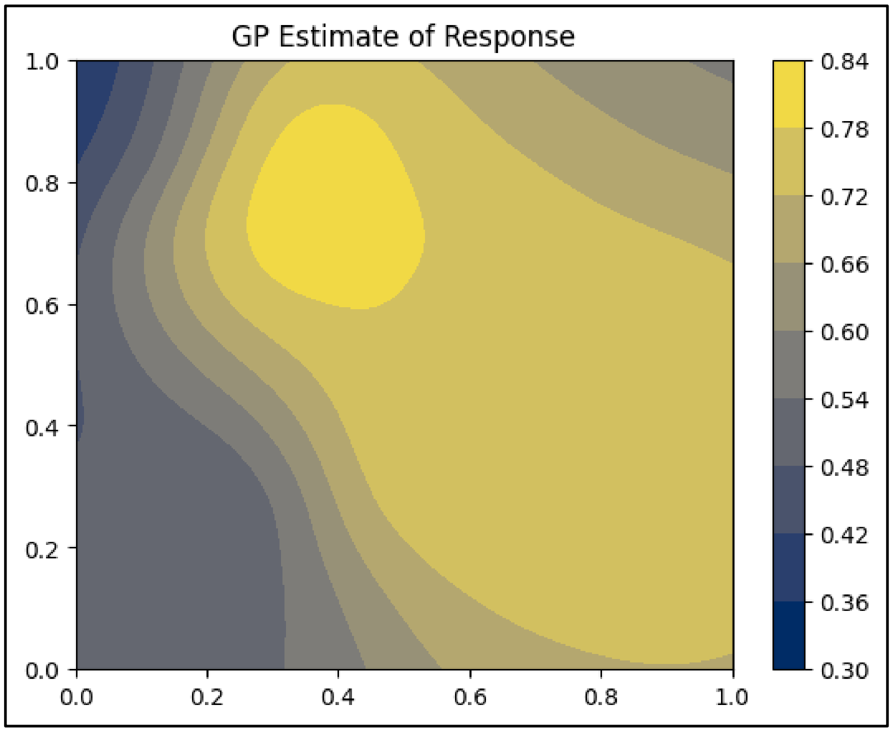


Figure 5: Gaussian Process Approximation of Response with RBF and Matern kernel

For the SVC experiment, we experiment with the *LHS* method by using an RBF and Matern kernel for its Gaussian surrogate. Comparing Figure 5 to Figure 4, it is evident that GP does provide a good approximation to the ground truth response surface.

V. Conclusion

Preliminary results demonstrated the potential of DoE techniques in modern ML. The Latin hypercube method exhibited notable competitiveness relative to advanced algorithms. Optimized space-filling in conjunction with GP modeling also captures response behavior and quantifies its uncertainty, making results more interpretable. Nonetheless, here GP is only used for analysis; further experiments of a true sequential approach with Bayesian optimization can be conducted. A downside of the LHS+GP combination is that it still relies on the optimal choice of kernels, which may require even more computational resources and domain knowledge.

The RSM implementation using the steepest ascent was comparably expensive and sub-optimal. Although the Plackett-Burman matrix can be utilized for cost-effective designs, scaling to a high dimension is still challenging for both first and second-order designs. Regardless, its performance is still highly dependent on the initialization of centers and step sizes. As a result, the efficacy of the steepest ascent is quite limited in HPO.

GitHub

https://github.com/mstrmnd1/hypercube

References

[1] Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019, July). Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining* (pp. 2623-2631).

[2] Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24.

[3] Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2).

[4] Feurer, M., Hutter, F. (2019). Hyperparameter Optimization. In: Hutter, F., Kotthoff, L., Vanschoren, J. (eds) *Automated Machine Learning*. The Springer Series on Challenges in Machine Learning. Springer, Cham. https://doi.org/10.1007/978-3-030-05318-5_1

[5] Frazier, P. I. (2018). A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811*.

[6] Seeger, M. (2004). Gaussian processes for machine learning. *International journal of neural systems, 14*(02), 69-106.

[7] Watanabe, S. (2023). Tree-structured Parzen estimator: Understanding its algorithm components and their roles for better empirical performance. *arXiv preprint arXiv:2304.11127*.

[8] Wu, C. J., & Hamada, M. S. (2011). *Experiments: planning, analysis, and optimization*. John Wiley & Sons.

Contribution

	Kaiwen Wang	Xiaochen Yan
Problem formulation	80%	20%
experiment	30%	70%
analysis	50%	50%
report	50%	50%
slides	50%	50%
overall	52%	48%