Monte Carlo Binary Chop for Hyper Parameter Optimization

George Kratz

March 2022

Abstract

Machine learning attempts to fit an objective function to data through optimization. Various machine learning algorithms exist to model complex data, many of which contain hyper-parameters, a set of meta-heuristics that guide the fitting of the algorithm. These hyper-parameters cannot be learned by the data, but must be fit prior to training. As a result, several algorithms have been developed to tune the hyper-parameters of these algorithms. We propose a simple algorithm, Monte Carlo Binary Chop, that across a variety of benchmarks outperforms both Gridsearch and Bayesian Optimization. We then introduce an early stopping criteria with a refined sampling algorithm and show that our algorithm is able to achieve better performance in less time than Bayesian Optimization.

1 Introduction

Machine learning attempts to fit an objective function to data through various optimization methods. While the actual method and objective function being fit vary based on model and modelling objective the underlying idea is constant. New variations in modelling techniques have developed over time leading to a rise in complexity, and the advent of the hyper-parameter.

Hyper-parameters are a set of meta-heuristics that guide the learning of the model. These hyper-parameters cannot be learned from the data, but must be chosen ahead of time. As a result, hyper-parameter selection has become a key tenet in model development and require either exhaustive trial and error, or domain expertise to narrow the selection of hyper-parameter values. This added layer of complexity has led to the advent of new meta-modelling and search techniques which attempt to aid in the selection of hyper-parameters. While well tuned hyper-parameters can provide an improved fit of the underlying model, the general complexity of selecting these parameters has increased.

Approaches to combating this complexity have been developed, from manual selection, or more complex genetic algorithms. The end result of each of these techniques is still the same, a tradeoff between time and fit to the data.

This paper aims to add to the arsenal of hyper-parameter selection techniques by proposing a new method, Monte Carlo Binary Chop. This method reduces the time complexity of hyper-parameter search, while maintaining flexibility in optimization goals, and allowing for complex or large hyper-parameter spaces to be searched, leading to cutting edge benchmark performance.

To support this claim, we first demonstrate the algorithm on a well documented dataset, and record benchmark performance. This performance is then demonstrated on additional datasets and compared to Bayesian optimization. Finally, we implement an intuitive early stopping criteria and refine the sampling algorithm, resulting in a significant decrease in optimization time without any loss in performance.

The rest of this paper is as follows: section 2 a literature review, section 3 an overview of the algorithm, section 4 demonstration of the algorithm and comparison to other methods, section 5 an early stopping criteria and Latin-Hypercube sampling, and section 6 conclusion.

2 Literature Review

The concept of monte-carlo search was first attributed to Stanislaw Ulam during his work at the Manhattan Project [Mar16]. While working on the Manhattan Project, Ulam needed a means to determine the patterns of neutron emission. While the basic patterns of the neutron motions where known, there existed no explicit solution to the problem.

After consideration, Ulam suggested that a numerical solution could be determined by repeatedly sampling the neutron's path, approximating the distribution. This proposed algorithm was termed Monte Carlo, named after the famous casino allegedly frequented by Ulam's Uncle [Bau].

The Monte Carlo method was subsequently adapted to solving optimization problems [Blu57][Cra55][Jon55]. Eventually leading to the advent of a core tool for Bayesian Statistics, the Markov Chain Monte Carlo [Met53].

The algorithm continued to have demonstrated success primarily in the fields of engineering. Where its prowess was demonstrated against more complex methods such as simulated annealing, genetic algorithms, and artificial bee colony in optimizing machining processes [Mad14].

Further research was conducted demonstrating the efficacy of Monte Carlo for solving simultaneous non-liner equations without the Newton-Rhapson method [Las12]. With further expansion into resolving Rufus Isaacs classical horse racing problem [Isa08] [Lash14], and other games such as Sudoku [Las15].

Some research has been conducted into random search for hyper-parameter optimization[Ber12]. Bergstra et al. find that random search can yield a first step in isolating which hyper-parameters are important for the given problem. Further research has been done into examining Monte Carlo Tree Search, where an approximation of the black-box optimization is used to compute potential performance on future samples [Raj18][Pol19]. This is an extension of a Monte Carlo search to optimizing the Multi-Armed Bandit problem.

A close relative to our proposed algorithm is the successive-halving[Jam15] or Hyperband[Lis16] algorithms for solving the Multi-Armed Bandit problem, which have recently been applied to hyper-parameter optimization. These algorithms impose a budgetary constraint on either time or performance, and allocate exponentially more resources to high performing hyper-parameter sets. However, our proposed algorithm removes the necessity of sampling every possible "Bandit", rather electing to sample randomly and shrink the search space towards higher-performing regions.

We aim to add to the repertoire of problems Monte Carlo methods have been able to solve. We demonstrate that Monte Carlo Binary Chop is effective at finding optimal hyper-parameters, and can be further improved with the introduction of early stopping and better sampling techniques.

3 Monte Carlo Binary Chop

3.1 Basic MCBC

Monte Carlo in its base form is a method of numerically approximating a function by random sampling. Traditionally, we see monte carlo used to approximate a derivative of a non-differential equation.

Given some function f with random variable x, and observed output y, by randomly drawing N observations over some region R of f we can obtain the integral of the function over that region:

$$\int_R f(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} f(x)$$

By extension, this then allows us to approximate the mean, $\bar{Y} = \mathbb{E}[Y] \approx \frac{1}{N}\sum_{i=1}^{N} f(x)$, and the variance, $\sigma_f^2 = \mathbb{E}[f^2] - \mathbb{E}[f]^2$. Where the variance of our estimated mean will decrease at a rate of $\sigma_f^2/\sqrt{N}$.

In numerical optimization, we would simply sample N times from some function $f(x)$, recording the best observed result. The benefit of random sampling is the ability of the function to avoid being constrained to local maxima or minima. The tradeoff is that we need a large N to ensure we are within some level of precision of the true maxima or minima, as the variance of our estimate will scale by a factor of $N^{-1/2}$.

The Monte Carlo Binary Chop (MCBC) algorithm is an extension of the traditional Monte Carlo whereby we stratify the samples N into I sized partitions. After I samples, we reduce the bounds of the random sample space by half, and repeat I additional samples until we have sampled N times total.

For optimizing hyper-parameters, we start the search by creating a fixed n-dimensional hyper rectangle around the hyper parameters. We then create a random set of hyper parameters that is a combination of the current upper and lower bounds with some random uniform shock. The model is then fit on the test array of hyper parameters and if the solution is improved, we store these values as the best found solution, recenter the hyper-rectangle, and continue the search.

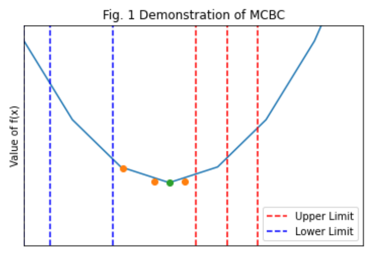
After repeating this search within the fixed bounds of the hyper rectangle I times, we reduce the bounds by half and repeat the search. The below pseudocode highlights this procedure.

Algorithm 1 MCBC Optimization

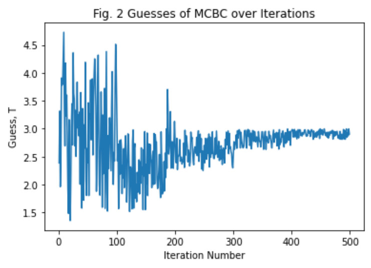
```
 1: Let LL, IW equal the lower limit, and interval width.
 2: Let A, T equal the best, and current iteration solutions found.
 3: Let F equal the best score.
 4: Let J, I equal the maximum power and iterations
 5: Initialize A = IW/2
 6: for $j = 0, 1, \ldots, J$ do
 7:     for $i = 1, 2, \ldots, I$ do
 8:         T = LL + (U(0,1)×IW)
 9:         Score the algorithm at parameters T
10:         if $score(T) > F$ then
11:             Set F = score(T) and A = T
12:             LL = $A - IW/2^j$
13:         end if
14:     end for
15:     IW = $IW/2^j$
16:     LL = $A - IW$
17: end for
```

It is trivial to continually add constraints to the algorithm, such as restricting one of the hyper parameters to only integers, or setting a minimum or maximum to the search space.

The below graphs shows a simple example of MCBC finding a global minimum of the function $y = (x-3)^2 + 5$.



(a) MCBC Minimization



(b) MCBC Guesses

Figure 1: Basic MCBC minimization

We can see that as the algorithm progresses and continues to take guesses, we repeatedly shrink the bounds of the search at each iteration until we converge to a global minimum in this case.

It should be noted that if insufficient iterations are allowed at each power, the algorithm will be unable to explore a sufficiently large space to determine the maximum of the optimization function.

In the case of the first halving, if the algorithm is not allowed to fully explore the space, and the space of the hyper parameter is sufficiently large, the algorithm will be unable to map out possible maximum points of the optimization function. As a result, the algorithm will get caught in a local rather than global maximum.

For these reasons, there should be a balance between speed and allowing the algorithm to fully map the space. Too many iterations and we begin to approximate a random search over the entire grid space. Too few iterations, and we will fail to determine a global maximum.

3.2 MCBC with Synthetic Data

We will explore a simple example of MCBC and traditional Monte Carlo tuning the single hyper-parameter of a lasso regression. We will tune the model over synthetic data, 10000 sample regression points, 10 informative features, and 90 non-informative features.

The Lasso regression model, and synthetic data are implemented using the scikit learn python library [SKLearn]. This implementation has one hyper-parameter, λ, with the objective function[Lasso]:

$$\sum_{i=1}^{n}(y_i - \sum_j x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \text{ where } \lambda \in \{0, 1\}$$

Where higher values of λ forces the non-informative covariates to zero.

We will test the two methods optimizing the mean squared error (MSE):

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2$$

We will test 100 iterations of each algorithm at total sample sizes, N, of 50, 100, 250. For the MCBC algorithm, the iterations will be split into 5 equal sized halves, i.e. 50 iterations becomes 5 halves of 10. The results in terms of MSE on a holdout validation set is reported below:

Table 1: MCBC vs Monte Carlo

	50 Iterations	100 Iterations	250 Iterations
	mean score	mean score	mean score
MCBC	0.0006 ± 0.0005	4.948e-05 ± 4.426e-05	3.113e-06 ± 3.663e-06
Monte Carlo	0.0088 ± 0.0185	0.00159 ± 0.0028	0.0003 ± 0.0004

μ is the average score, and σ^2 is the standard deviation of score.

It is very clear that MCBC is vastly superior in terms of optimization than Monte Carlo search. In fact, the performance with 50 iterations is over a full order of magnitude better, while both 100 iterations and 250 iterations are roughly two orders of magnitude better than the random search.

4 Demonstration

For demonstrating the effectiveness of the MCBC algorithm, we will first use the non-early stopping MCBC algorithm to highlight the technique. Then across an additional 5 datasets we will compare the performacne of MCBC both with and without early stopping against the Bayesian Optimization algorithm.

4.1 XGBoost Example

The Extreme Gradient Boosted model, XGBoost, is an ensemble algorithm that improves upon the gradient boosted tree algorithm.

While the full mathematical formulation of gradient boosted trees and XGBoost is outside the scope of this research, the author has included the original XGBoost paper in the bibliography [Chen16].

For this example, we will consider a case where we are attempting to optimize 4 hyper-parameters, denoted in the below table:

Table 2: XGBoost Hyper-Parameters

Parameter	Bounds	Type
eta	$\in (0, 1)$	Float
max depth	$\in (0, \infty)$	Integer
subsample	$\in (0, 1)$	Float
colsample_bytree	$\in (0, 1)$	Float

Assuming we were to test 10 values within each space, we would be exploring a search space of 10000 possible combinations. In comparison, the proposed monte-carlo algorithm always runs in constant time. We can fully explore this space in significantly less time with a more conservative 5 chops, with 300 iterations at each halving. Given a more complex search space, we would appropriately increase the random search space by increasing the number of iterations at each halving.

In this example, we will attempt to tune these 4 hyper-parameters across the census income dataset.

4.2 Census Income Dataset

The purpose of this dataset is to categorize individuals into above and below $50k income. It is extracted from the 1994 census data. The data were provided by the UCI Machine Learning Dataset Archives [ADU96].

The panel data were split into a training and test set, continuous variables mean, standard deviations, and difference in mean p-values are denoted below:

Table 3: Descriptive Statistics of Continuous Variables

Variable	Train			Test			P-Value
	Count	Mean	SD	Count	Mean	SD	
Age	32560	38.58	13.64	16281	38.77	13.85	0.158
Fnlwgt	32560	189781.80	105549.80	16281	189435.70	105714.90	0.733
Education	32560	10.08	2.57	16281	10.07	2.57	0.756
Capital-gain	32560	1077.62	7385.40	16281	1081.91	7583.94	0.952
Capital Loss	32560	87.31	402.97	16281	87.90	403.11	0.878
Target Variable	32560	0.241	0.428	16281	0.236	0.425	0.262

Two-tailed P-Values of difference in means, significant values denoted with *.

Table 4: Categorical Variables For Income Census Data

Variable	Train		Test	
	Most Frequent	Total Categories	Most Frequent	Total Categories
WorkClass	Private	9	Private	9
Marital Status	Married-civ-spouse	7	Married-civ-spouse	7
Occupation	Prof-specialty	15	Prof-specialty	15
Relationship	Husband	6	Husband	6
Race	White	5	White	5
Sex	Male	2	Male	2
Native Country	United-States	42	United-States	41*

The missing value for Native Country in the test set is Holland-Netherlands. There is only one instance of this category in the training dataset, so this instance was dropped from the training dataset.

Categorical variables were numerically encoded from 0 to number of categories - 1. No other data-transformations or data cleaning was performed on the test and train datasets. The final datasets were comprised of 13 variables, 32559 instances in the training set, and 16281 in the test set.

The UCI Machine Learning Archives display benchmark scores by model type for the data. The benchmark score for this dataset with XGBoost was, as far as the author can find, achieved by Topiwalla with an accuracy score of 87.53% [Top].

To test the ability of the MCBC algorithm to tune the hyperparameters of XGBoost, we will conduct 10-fold crossvalidation on the training dataset, with 5 binary chops, 300 iterations per chop. It should be noted that Topiwalla performed some missing value inference, and tested over a smaller test set, 9769 instances compared to 16281 in the full dataset [Top]. To make the tests more comparative, we will run a second test across the dataset used by Topiwalla [Top].

All tests were conducted on a machine with a processor Intel®Core™I7 1165G7 CPU, 2.8 GHz, with 16.0 GB memory.

The below chart shows the evolution of the model performance over training iterations for both the full dataset, and the Topiwalla dataset:

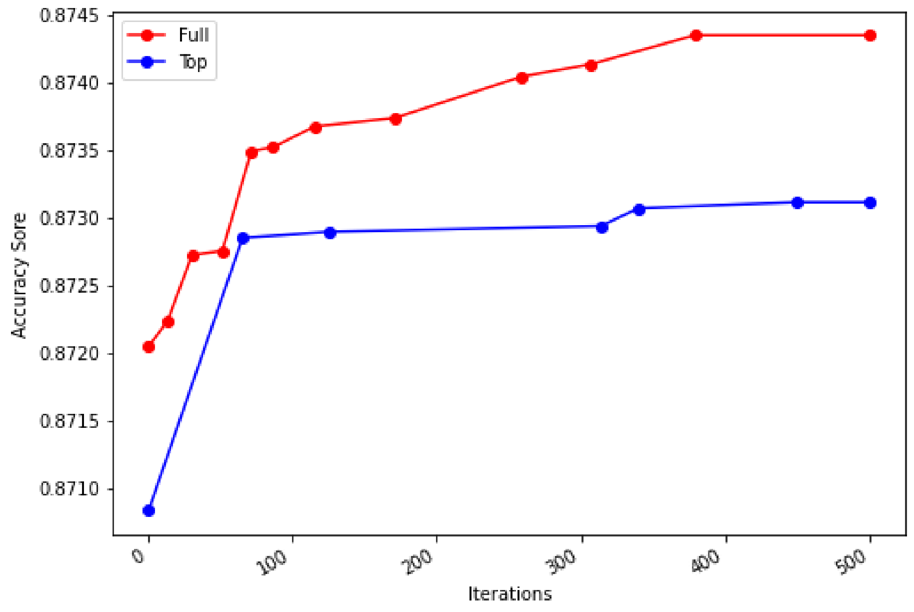


Figure 2: MCBC Census Training Score

The final scores for the model on both validation datasets are reported below:

It should be noted that the MCBC algorithm was able to achieve greater performance in all 3 reported metrics over the Topiwalla dataset. Additionally,

Table 5: Performance on Census Income Dataset

Dataset	Accuracy	ROC-AUC Score	Cohen-Kappa
MCBC Full	87.45%	0.9278	0.6328
MCBC Topiwalla	87.57%	0.9306	0.6454
Topiwalla Bencmark	87.53%	0.9275	0.6326

the performance across the full dataset is above the median score as reported by UCI [ADU96].

4.3 Additional Datasets

To Further illustrate the performance of this hyper-parameter search algorithm, we introduce an additional 5 benchmark datasets, and compare the performance between Bayesian Optimization, and the MCBC algorithm.[1]

4.3.1 Bayesian Optimization

Bayesian Optimization is regarded as one of the leading optimization algorithms. As denoted by Yang et al.'s analysis of the current state of hyper-parameter optimization methods, Bayesian optimization is often both one of the fastest, and best performing algorithms across a wide range of problems[Yan20].

Bayesian Optimization models the black-box function of the model performance by a Gaussian Process:

$$P(f|D_{1:t}) \propto L(D_{1:t}|f)\pi(f) \tag{1}$$

Where

$$f = GP(m(x), K(x, x^{'})) + \epsilon$$

With priors on m(x) and $K(x, x^{'})$):

$$\pi(m(x)) = 0$$
$$\pi(k(x, x^{'})) = exp(-\tfrac{1}{2}\|x_i - x_j\|^2)$$
$$\pi(\epsilon) = N(0, \sigma^2)$$

Where m(x), $k(x, x^{'})$ are the mean and covariance function of the unknown function f(x) modelled by a Gaussian Process, and $D_{1:t}$ is the observed data. Where m(x) is assumed to have 0 mean. The choice in covariance function can also vary, but for this overview, we model the covariance as a squared exponential function[Bro10]. While ϵ is the error of the Gaussian Process. This

[1]Car Evluation Dataset [CAR97], Nursery Dataset[NUR97], Optical Recognition of Handwritten Digits[OPT98], Spambase Dataset[SPA99], Image Segmentation Dataset[IMA90]

gaussian process has the nice property of modelling each sample as a multivariate normal distribution.

We then use an acquisition function to determine where our next sample will be drawn from. The choice of acquisition function can vary, but for the sake of explanation, we will discuss the expected improvement function. The expected improvement function selects the next sample as the sample that provides the greatest expected improvement:

$$x_t = argmin\ \mathbb{E}(\|f_{t+1}(x) - f(x^*)\|| D_{t+1}) \tag{2}$$

Where x_{t+1} is the proposed new point, $f(x^*)$ is the best observed performance of the GP. It should be noted that this relationship must be strictly positive and it is common to see the Expected Improvement rewritten as[Bro10]:

$$x_t = argmin\ \mathbb{E}(max(f_{t+1}(x) - f(x^*), 0)|D_{t+1})$$

In this base context, the Bayesian Optimization algorithm pseudocode is as follows:

Algorithm 2 Bayesian Optimization

```
1: Sample n random points in the space X
2: Update $D_{1:t}$ with sample X
3: Update GP, Equation 1, with $D_{1:t}$
4: for 1, 2, ..., t do
5:     Find $X_t$ by Optimizing the acquisition function, Equation 2, over the GP, Equation 1
6:     Sample the GP, Equation 1, over the new point $X_t$, $y_t = GP(X_t)$
7:     Add $X_t$, $y_t$ to the data,$D_{1:t}$
8:     Update the GP, Equation 1, with the new data
9: end for
```

More on this can be found in the synopsis by Brochu et al. [Bro10]. Brochu et al. provides additional oversight on the basis of Bayesian Optimization as well as additional coverage of choice of covariance functions, and acquisition functions.

We have chosen to compare the performance of MCBC to Bayesian Optimization (Bayes Opt), as Bayes Opt has been demonstrated to achieve benchmark performance in significantly less time than more exhaustive methods [Wu19], and should prove a good benchmark for the MCBC algorithm. Bayesian Optimization for each benchmark was performed by the Scikit-Optimization library in python [Skopt].

4.3.2 Performance Comparison

For the additional datasets, we use the RandomForest algorithm.

The RandomForest algorithm has two parameters we will tune, the number of estimators and the maximum depth of the tree. Both algorithms will be optimizing the number of estimators in the range $\in \{1, 200\}$ and maximum depth in the range $\in \{1, 20\}$.

For each of these datasets, we conduct 5 binary chops, and 10 iterations at each chop, for a total of 50 samples. Bayes Opt will also be allowed 50 samples from the search space. Additionally, each dataset will be split into a 75-25 training and validation set. 10-fold cross validation will be performed on the training split, and the parameters with the highest average accuracy score across each 10-folds will be retained as the parameters for scoring on the validation dataset.

For each dataset no feature engineering was conducted. Categorical variables were numerically encoded as 0 to number of categories -1.

Table 6: Description of Datasets

Dataset	Samples	Attributes	Attribute Type	Task
Car Evaluation	1728	6	Categorical	Multi-Class Classification
Nursery	12960	8	Categorical	Multi-Class Classification
Optical Digits	5620	64	Real	Multi-Class Classification
Spambase	4601	57	Real	Classification
Image Segmentation	2310	19	Real	Multi-Class Classification

All datasets were provided by the UCI machine learning archives,

For performance comparisons, since both algorithms are stochastic, we run 10 iterations of each, averaging the time of each run, and the score. For ease of comparison, we will introduce two additional metrics, RYC, relative test score change, and RTC, relative test time change, [Mak22] defined as follows:

$$RYC = \frac{y_M - y_B}{max(y_M, y_B)} \text{ where } RYC \in \{-1, 1\}$$

$$RTC = \frac{t_B - t_M}{max(t_B, t_M)} \text{ where } RTC \in \{-1, 1\}$$

Where y_M and y_B are the scores of the MCBC and Bayes algorithms respectively, and t_M, t_B are the times of the two algorithms respectively. A positive RYC score denotes an increase in performance relative to the alternative algorithm, while the RTC score denotes the reduction in training time between the two tests. The below table denotes the performance of both algorithms.

Table 7: MCBC vs Bayesian Optimization

	MCBC		Bayes		Comparison	
Dataset	Score (%)	Time (S)	Score (%)	Time (S)	RYC (%)	RTC (%)
Car Evaluation	97.52	131	97.59	113	-0.07	-13.74
Nursery	98.47	295	98.45	243	0.02	-17.63
Optical Digits	97.73	253	97.69	195	0.04	-22.92
Spambase	95.35	166	95.32	199	0.03	16.58
Image Segmentation	94.93	186	94.90	170	0.03	-8.60

Across the benchmark datasets, the MCBC algorithm's average performance was above the Bayes Optimization algorithm on all but one dataset. However, the MCBC algorithm was on average 9% slower than Bayesian Optimization.

5 Improving the MCBC

We propose three adaptations to the MCBC algorithm. If we break the algorithm into three components, we have an initial exploration phase, an acquisition/optimization phase, and a stopping criteria. To improve the algorithm, we suggest a more efficient exploration search method, an improved acquisition/optimization method, and an intuitive early stopping criteria.

5.1 Exploration Phase

The initial pass through the search space before reducing bounds should ideally fully capture or atleast represent potential maxima or minima within the search space. Random search, the traditional MCBC method, does not guarantee full coverage of the search space.

Therefore, we propose a modification of the algorithm where we draw the samples such that each sample is the only sample in each axis aligned hyperplane.

This method is known as latin-hypercube sampling (LHS)[McK79]. LHS guarantees the samples are representative of the true variability of the space. The method is fairly straightforward, let X_d^N be a d dimensional space of N samples. The LHS sample is then to separate the space X_d^N into M strata with equal marginal probability, $1/M$.

However, some studies[Dam13] have found that traditional LHS sampling fails to properly fill the space, and so we will instead draw samples such that:

$$\phi M_m(X_d^N) = \min_{\substack{i,j=1,..N \\ i \neq j}} d_{ij} \tag{3}$$

Where

$$d_{ij} = \|x_i - x_j\|$$

Thus, d_{ij} is the distance between points i and j. We therefore find the points which maximizes $\phi M_m(X_d^N)$ and minimizes the minimal distance.

These samples are also guaranteed to be representative of the true variance of the distribution, and are therefore guaranteed to have a better representation of the variance of the search space than random sampling. This guarantee allows us to use the initial exploration phase with Minimax LHS to estimate our priors over the search space, which will be used for our early termination criteria.

5.2 Acquisition Criteria

The current acquisition method is to repeatedly move towards the region that contains the highest score.

However, as suggested by Cox et al's [Cox92][Cox97] work on globabl optimization, we instead will select points to further explore based on their upper confidence bound.

In theory, the k-fold cross validated estimate of performance is an unbiased representation of the population performance of the set of hyper-parameters[Bay20]. However, the variance of k-fold crossvalidation is generally unknown, but can be approximated with a simple correction [Nad99] based on the number of folds:

$$Var_{cv} = (\tfrac{1}{k} + \tfrac{1}{k-1})\sigma_{cv}^2$$

Therefore, by means of the k-fold cross validation performance, we can estimate the upper confidence bound of the population performance of the hyper-parameters by:

$$UCB_{cv} = \mu_{cv} + \sqrt{\beta}\sqrt{Var_{cv}} \tag{4}$$

Where β is the number of hyper-parameters we are tuning as suggested by Makarova et al. [Mak22].

Using this specification, we will instead move towards the solution that contains the greatest cross-validated upper confidence bound, which is theoretically the best performing population hyper-parameters.

5.3 Termination Criteria

In the original MCBC, after randomly sampling I times from the current region, we then recenter the bounds around the best found solution, reduce the bounds by half, and sample an additional I times from the new region.

By centering the new search region around the best found solution, we are assuming that further exploration of that region will yield greater performance.

There is no guarantee that this new region will provide a more optimal solution than the current found solution.

Therefore, we propose a modification to the original algorithm by terminating the search once the potential for improvement, as measured by the upper or lower confidence bounds, is below the current best score.

The estimation of the confidence bound is according to the posterior μ and σ estimates. Given the small sample number of samples, there is no guarantee the samples will approximate a normal distribution.

However, Bayesian posterior inference allows us to overcome the sampling limitations. Rather than rely on a frequentist estimate of the population distribution, we can take advantage of prior observations from LHS to gain a more accurate estimation of the population distribution.

We use the samples obtained in the initial exploration phase as our priors for μ and σ. The specification for the posterior inference is below:

$$P(\mu, \sigma | x) \propto L(N(\mu, \sigma)) \times \pi(\mu) \times \pi(\sigma) \tag{5}$$

With priors,

$$\begin{gathered} \pi(\mu) = N(\mu, \sigma) \\ \pi(\sigma) = HN(\sigma) \end{gathered}$$

In this specification, we assume the prior performance on μ is Normally distributed, and with σ being half-normally distributed. The specification of a half-normal prior on σ is due to the variance of the performance assumed to be closer to zero, this prior then is weakly informative, and restricts σ away from large values.

We sample I times at each halving, and reduce the bounds by half. For the best observed hyper-parameter set, we determine the upper-confidence region by 5 and estimate the posterior μ and σ by means of Hamiltonian Monte Carlo No U-Turn Sampling which was achieved by use of the Pymc3 library in python [Pymc3]. We then use this posterior estimate to calculate the upper confidence bound as defined by:

$$UCB = \mu + \sqrt{\beta}\sigma \tag{6}$$

We update our prior belief about possible performance in this region by setting the priors as the posterior estimates of μ and σ

If at any point, the new region exhibits a lower posterior UCB than the current best score, we terminate the search.

The full pseudocode for the early stopping algorithm is below:

Algorithm 3 MCBC UCB Optimization

```
1: Let LL, UL, IW equal the lower limit, upper limit, and interval width.
2: Let A, T equal the best, and current iteration solutions found.
3: Let F equal the best score.
4: Let J, I equal the maximum power and iterations
5: Let $\pi(\mu)$, $\pi(\sigma)$ be initial priors for mean and variance.
6: Initialize A = IW/2
7: for $j = 1, \dots, J$ do
8:     if j = 1 then
9:         Find the set T of length I by equation 3, $\in \{LL, UL\}$
10:        for t in T do
11:            Score the algorithm at parameters t according to 4
12:            if $ucb_{cv}(t) > F$ then
13:                Set F = $ucb_{cv}(t)$ and A = t
14:            end if
15:        end for
16:        Initialize priors over the observed scores $\pi(\mu)$ and $\pi(sigma)$
17:    else
18:        Find the set T of length I by sampling randomly within the bounds $\in \{LL, UL\}$
19:        for t in T do
20:            Score the algorithm at parameters t according to 4
21:            if $ucb_{cv}(t) > F$ then
22:                Set F = $ucb_{cv}(t)$ and A = t
23:                Recenter bounds around the new best solution
24:            end if
25:        end for
26:    end if
27:    IW = $IW/2^j$
28:    UL = $A + IW$
29:    LL = $A - IW$
30:    Calculate the UCB, 6, of the new region, and set the prior $\pi(\mu)$ and $\pi(\sigma)$ for the region to be the posterior $\mu$ and $\sigma$.
31:    if UCB < F then
32:        Terminate
33:    end if
34: end for
```

5.4 Upper Confidence Bound MCBC Performance

For each dataset we will allow the MCBC algorithm to sample 10 points at each halving, with 5 total halvings, stopping when either the UCB is below the best reported score, or 50 iterations have elapsed. For the Bayesian optimization, we will run 50 iterations. Therefore, both algorithms will be allowed to run a maximum of 50 times, and any difference in time would be the result of the early stopping criteria terminating the search for the MCBC algorithm.

We will run both algorithms through the datasets 10 times each, recording the average search time, average accuracy score, and standard deviation of score on the holdout validation dataset.

The reported performance across each dataset for each algorithm is below:

Table 8: MCBC vs Bayesian Optimization

	MCBC w/ UCB		Bayes Opt		Comparison	
Dataset	Score (%)	Time (S)	Score (%)	Time (S)	RYC (%)	RTC (%)
Car Evaluation	97.52	29	97.59	113	-0.071	74.33
Nursery	98.45	89	98.45	243	0.000	63.37
Optical Digits	97.85	94	97.69	195	0.164	54.36
Spambase	95.25	78	95.32	199	-0.070	60.80
Image Segmentation	94.79	80	94.90	170	-0.116	52.94

The reported score is the accuracy on a hold-out validation dataset.

The MCBC algorithm with early stopping was able to achieve a 62% reduction in computation time, with a roughly 2% decrease in performance.

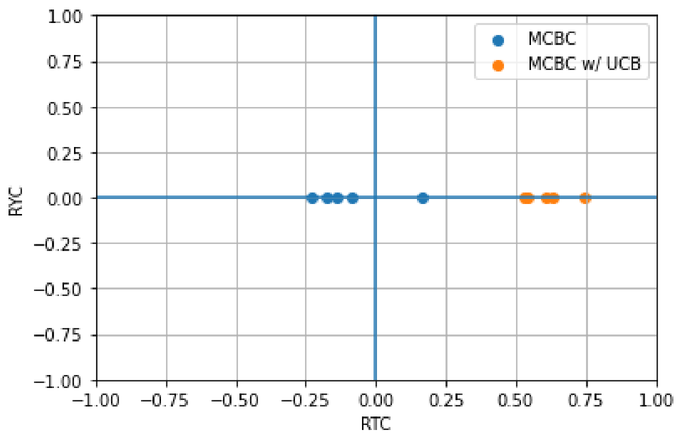


Figure 3: Comparison MCBC and MCBC w/ UCB

The final comparison of the MCBC with early stopping to the original MCBC shows the significant time increase of MCBC with early stopping over MCBC. Furthermore, there is no significant decrease in performance across the datasets.

6 Conclusion

Throughout this paper, we have demonstrated that a simple algorithm is capable of achieving competitive, and benchmark performance by optimizing the hyper-parameters of various algorithms. Across multiple datasets, we have demonstrated the MCBC algorithm is capable of finding an optimal set of hyper-parameters in a small number of iterations. Additionally, we have demonstrated that the algorithm is capable of tuning a large number of hyper-parameters simultaneously. Finally, we have demonstrated the MCBC algorithm with a simple early stopping criteria and a refined search algorithm is capable of significant computational cost reduction without a loss in performance.

Bibliography

[ADU96] *Adult.* UCI Machine Learning Repository. 1996.

[OPT98] E. Alpaydin and C. Kaynak. *Optical Recognition of Handwritten Digits.* UCI Machine Learning Repository. 1998.

[Bau] W. F. Bauer. "The Monte Carlo Method". In: *Journal of the Society for Industrial and Applied Mathematics* 6 (), pp. 438–451.

[Bay20] Pierre Bayle et al. "Cross-validation Confidence Intervals for Test Error". In: *Advances in Neural Information Processing Systems.* Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 16339–16350. URL: https://proceedings.neurips.cc/paper/2020/file/bce9abf229ffd7e570818476ee5d7dde-Paper.pdf.

[Ber12] James Bergstra and Yoshua Bengio. "Random Search for Hyper-Parameter Optimization". In: *J. Mach. Learn. Res.* 13.null (Feb. 2012), pp. 281–305. ISSN: 1532-4435.

[Blu57] A. Blumstein. "A Monte Carlo analysis of the Ground Controlled Approach system." In: *Operations Research* 5 (1957), pp. 397–408.

[Bro10] Eric Brochu, Vlad Cora, and Nando Freitas. "A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning". In: *CoRR* abs/1012.2599 (Dec. 2010).

[SKLearn] Lars Buitinck et al. "API design for machine learning software: experiences from the scikit-learn project". In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning.* 2013, pp. 108–122.

[CAR97] *Car Evaluation.* UCI Machine Learning Repository. 1997.

[Chen16] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *CoRR* abs/1603.02754 (2016). arXiv: 1603.02754. URL: http://arxiv.org/abs/1603.02754.

[Cox92] Dennis D. Cox and Susan John. "A statistical method for global opti- mization". In: *IEEE Conference on Systems, Man and Cybernetics.* 1992.

[Cox97] Dennis D. Cox and Susan John. "SDO: A Statistical Method for Global Optimization". In: *in Multidisciplinary Design Optimization: State-of-the-Art.* 1997, pp. 315–329.

[Cra55] R. R. Crane, F.B. Brown, and R. O. Blanchard. "An analysis of a railroad classification yard." In: *Journal of the Operations Research Society of America.* 3 (1955), pp. 262–271.

[Dam13] G Damblin, M Couplet, and B Iooss. "Numerical studies of space-filling designs: optimization of Latin Hypercube Samples and subprojection properties". In: *Journal of Simulation* 7.4 (2013), pp. 276–289. DOI: 10.1057/jos.2013.16. eprint: https://doi.org/10.1057/jos.2013.16. URL: https://doi.org/10.1057/jos.2013.16.

[Lasso] Stephanie Glen. *Lasso regression: Simple definition*. Apr. 2021. URL: https://www.statisticshowto.com/lasso-regression/.

[Skopt] Tim Head et al. *scikit-optimize/scikit-optimize*. Version v0.8.1. Sept. 2020. DOI: 10.5281/zenodo.4014775. URL: https://doi.org/10.5281/zenodo.4014775.

[SPA99] Hopkins et al. *Spambase*. UCI Machine Learning Repository. 1999.

[IMA90] *Image Segmentation*. UCI Machine Learning Repository. 1990.

[Isa08] Rufus Isaacs. "OPTIMAL HORSE RACE BETS". In: *Efficiency Of Racetrack Betting Markets*. World Scientific Publishing Co. Pte. Ltd., 2008. Chap. 13, pp. 93–97. URL: https://EconPapers.repec.org/RePEc:wsi:wschap:9789812819192_0013.

[Jam15] Kevin G. Jamieson and Ameet Talwalkar. "Non-stochastic Best Arm Identification and Hyperparameter Optimization". In: *CoRR* abs/1502.07943 (2015). arXiv: 1502.07943. URL: http://arxiv.org/abs/1502.07943.

[Jon55] H. G. Jones and A. M. Lee. "Monte Carlo methods in heavy industry". In: *Operational Research Quarterly.* 6 (1955), pp. 108–116.

[Lash14] Jacob Lashover. "A Chemical Engineer Goes to the Horse Races". In: (Aug. 2014).

[Las15] Jacob Lashover. "Mathematical Solution to Sudoku Using Monte Carlo Marching". In: (Sept. 2015).

[Las12] Jacob Lashover. "Monte Carlo Marching". In: (Nov. 2012).

[Lis16] Lisha Li et al. "Efficient Hyperparameter Optimization and Infinitely Many Armed Bandits". In: *CoRR* abs/1603.06560 (2016). arXiv: 1603.06560. URL: http://arxiv.org/abs/1603.06560.

[Mad14] Miloš Madić, Marko Kovačević, and Miroslav Radovanović. "Application of multi-stage Monte Carlo method for solving machining optimization problems". In: *International Journal of Industrial Engineering Computations* 5 (Aug. 2014), pp. 647–659. DOI: 10.5267/j.ijiec.2014.7.002.

[Mak22] Anastasia Makarova et al. *Automatic Termination for Hyperparameter Optimization*. 2022. URL: https://openreview.net/forum?id=2NqIV8dzR7N.

[Mar16] Robert Marks. "Monte Carlo". In: Jan. 2016, pp. 1–4. DOI: 10.1057/978-1-349-94848-2_709-1.

[McK79] M. D. McKay, R. J. Beckman, and W. J. Conover. “A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code”. In: *Technometrics* 21.2 (1979), pp. 239–245. ISSN: 00401706. URL: `http://www.jstor.org/stable/1268522`.

[Met53] Nicholas Metropolis et al. “Equation of State Calculations by Fast Computing Machines”. In: *The Journal of Chemical Physics* 21.6 (1953), pp. 1087–1092. DOI: `10.1063/1.1699114`. eprint: `https://doi.org/10.1063/1.1699114`. URL: `https://doi.org/10.1063/1.1699114`.

[Nad99] Claude Nadeau and Yoshua Bengio. “Inference for the Generalization Error”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Solla, T. Leen, and K. Müller. Vol. 12. MIT Press, 1999. URL: `https://proceedings.neurips.cc/paper/1999/file/7d12b66d3df6af8d429c1a357d8b9e1a-Paper.pdf`.

[Pol19] Karolina Polanska et al. “Monte Carlo Tree Search for Optimizing Hyperparameters of Neural Network Training”. In: *Icons 2019: The Fourteenth International Conference on Systems*. 2019.

[NUR97] Vladislav Rajkovic. *Nursery*. UCI Machine Learning Repository. 1997.

[Pymc3] John Salvatier, Thomas V Wiecki, and Christopher Fonnesbeck. “Probabilistic programming in Python using PyMC3”. In: *PeerJ Computer Science* 2 (2016), e55.

[Raj18] Rajat Sen, Kirthevasan Kandasamy, and Sanjay Shakkottai. “Noisy Blackbox Optimization using Multi-fidelity Queries: A Tree Search Approach”. In: *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*. Ed. by Kamalika Chaudhuri and Masashi Sugiyama. Vol. 89. Proceedings of Machine Learning Research. PMLR, Apr. 2019, pp. 2096–2105. URL: `https://proceedings.mlr.press/v89/sen19a.html`.

[Top] Mohammed Topiwalla. “Machine Learning on UCI Adult data Set Using Various Classifier Algorithms And Scaling Up The Accuracy Using Extreme Gradient Boosting”. PhD thesis.

[Wu19] Jia Wu et al. “Hyperparameter Optimization for Machine Learning Models Based on Bayesian Optimization”. In: *Journal of Electronic Science and Technology* 17.1 (2019), pp. 26–40. ISSN: 1674-862X. DOI: `https://doi.org/10.11989/JEST.1674-862X.80904120`. URL: `https://www.sciencedirect.com/science/article/pii/S1674862X19300047`.

[Yan20] Li Yang and Abdallah Shami. “On hyperparameter optimization of machine learning algorithms: Theory and practice”. In: *Neurocomputing* 415 (2020), pp. 295–316. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2020.07.061. URL: https://www.sciencedirect.com/science/article/pii/S0925231220311693.