ADBA Computer Science
Cutting-Edge Scientific Solutions

e-ISSN: 3023-8544
RESEARCH ARTICLE
Vol.3 / No.1 / 2026 / pp.6-12
https://doi.org/10.69882/adba.cs.2026012

Hyperparameter Optimization for Big Data: Adapting Sampling Methods to Apache Spark MLlib

M. Maruf Öztürk *,[1]
*Department of Computer Engineering, Engineering and Natural Sciences Faculty, Suleyman Demirel University, Isparta, Türkiye.

ABSTRACT MLlib is an Apache Spark library that provides many machine learning algorithms and data processing utilities. Although the default configuration of these algorithms yields satisfactory results for practitioners, further tuning is often needed to improve resource usage efficiency. Furthermore, tuned MLlib algorithms may run faster than those using default configurations. However, this improvement depends on several factors, including machine settings, dataset design, and operating system preferences. Previous studies have generally focused on developing sophisticated tuners for MLlib, evaluating algorithm-focused optimizers for their competitiveness. Although derivative-based and model-free optimizers have been modified for use with MLlib, sampling-based optimizers are generally overlooked. To fill this research gap, this study empirically compares sampling-based and model-free techniques for tuning MLlib. Firstly, Monte Carlo and Cross-Entropy sampling algorithms are adapted to optimize MLlib algorithms. Subsequently, model-free techniques, including grid and random search algorithms, are compared with these sampling-based algorithms. Through extensive experimentation, their advantages and limitations are highlighted. Finally, threats to validity and future directions for unlocking the tuning potential of Apache Spark are discussed by interpreting performance bottlenecks and promising areas for optimization.

KEYWORDS
Optimization
Apache spark
Tuning
Monte carlo
Cross-entropy

INTRODUCTION

Hyperparameter optimization (HO) is an indispensable yet intricate process in machine learning (ML) (Feurer and Hutter 2019). For this reason, several HO techniques exist, including random (Khaldi et al. 2025) and grid search (Ilmeboya et al. 2024) algorithms, Bayesian optimization (Eleftheriadis et al. 2024), the Nelder-Mead method (Herbst et al. 2024), and other heuristics (van Stein et al. 2024). Model-free HO techniques, such as grid search, are commonly embedded into automated ML (AutoML) systems. This integration facilitates the entire HO process by enabling a discussion of the advantages and disadvantages of existing alternatives.

Apache Spark is a big data processing framework with over 200 configurable hyperparameters (Meng et al. 2016). To achieve maximum efficiency, the default configuration of these hyperparameters should be compared with optimized ones. However, selecting the right optimization algorithm is a complex and effort-intensive process. This complexity arises because the Resilient Distributed Dataset (RDD), a core component of Apache Spark, requires a significant amount of time to solve even a moderate optimization problem. To address this issue, preliminary experiments are often performed to predict overall performance. Moreover, the parallel execution of hyperparameter searches can shorten the optimization process (Meister et al. 2020).

MLlib was originally designed for performing machine learning tasks, including feature selection, classification, regression, and clustering (Assefi et al. 2017). Although it yields satisfactory results with the default hyperparameters proposed by Apache Spark, resource management issues such as CPU usage, execution time, and memory consumption can often only be controlled by configuring MLlib's hyperparameters. For model-free tuning, random search may be a good solution when the tuning budget is limited. Although such techniques have achieved significant reductions in execution time, their performance depends greatly on the sampling conducted within the local search space (Nguyen et al. 2018). The number of possible configurations is very large, so any comprehensive search technique may require several days to find an optimal set of parameters. Reducing this search space may be possible by using a Bayesian algorithm to eliminate redundant candidate points (Cao et al. 2024).

Cross-entropy (CE) is a sampling-based minimization algorithm consisting of two main phases: generating random instances and updating model parameters (Benham et al. 2017). It has been applied to various optimization problems, including manufacturing (Beruvides et al. 2016), sensor management (Naeem et al. 2009), engineering design (Li et al. 2019), and path planning (Tang et al. 2024). CE is very effective in landscapes with many local optima, such as those found when optimizing Apache Spark.

Monte Carlo is a sampling-based optimization algorithm that predicts possible outcomes by repeating simulations. It leverages probability distributions to produce outputs. In particular, the Hamiltonian version of Monte Carlo is suitable for complex, real-world optimization problems (Campbell et al. 2021). Note that the scale factor can significantly improve the general performance of the Monte Carlo method.

Manuscript received: 12 December 2025,
Revised: 13 January 2026,
Accepted: 13 January 2026.

[1]muhammedozturk@sdu.edu.tr (**Corresponding author**)

Although each HO technique has distinctive characteristics that can be observed during optimization (Andonie 2019), big data processing platforms require sophisticated libraries to perform tuning. For instance, Apache Spark MLlib only supports cross-validation and the generation of a parameter grid for hyperparameters (Zhu et al. 2025). Therefore, the tuning process becomes computationally expensive due to data-based parallelism (Zhou et al. 2025). Consequently, there is a need to perform HO with techniques compatible with RDDs. This approach can remarkably reduce the time allocated for HO. Alternatively, custom-tuned models can be used to replace the default algorithms provided by MLlib.

Despite the common use of model-free HO techniques, it remains unclear whether sampling-based methods, such as Monte Carlo and cross-entropy, are superior. To bridge this gap, we perform a comprehensive comparison of model-free and sampling-based algorithms. To this end, four classification datasets are processed using MLlib algorithms alongside hyperparameter sets proposed by the comparison algorithms. The convergence, execution time, and rejection rate of four optimization algorithms Monte Carlo, grid search, random search, and CE were evaluated.

The paper claims the contributions as follows: 1) We investigate Apache MLlib whether it is compatible with simulation-based optimization, 2) We compare Monte Carlo with traditional techniques to reveal the most suitable hyperparameter optimization technique for Apache Spark, 3) We present a comprehensive complexity analysis of the baselines for the practitioners. The remainder of the paper is organized as follows: Section 2 gives definitions of the background. Section 3 summarizes related work. Section 4 elucidates the proposed method. The results are presented in Section 5. Threats to validity are discussed in Section 6. Finally, Section 7 concludes the paper and summarizes the results.

BACKGROUND

Monte Carlo expectation: Monte Carlo runs for approximating a random process in a specific number of iteration. A good estimation can be calculated as $\theta_n = \sum_{i=1}^{n} P_i/n$ in which n is the number of points and P is a candidate point.

Cross-entropy minimizer: Let S be a set of states and f is the performance function used in S. The main aim is to minimize f over S in which a corresponding minimizer $c*$ is responsible for the success of f. ($f(c*) = min_{f(X)}, X \in S$)

Cross-entropy convergence: Convergence of cross-entropy can be calculated with a chain rule that utilizes some partial derivations: $\frac{\partial L}{\partial w} = (-y(1-p) + (1-y)p) \cdot x$ $\frac{\partial L}{\partial w} = (-y + yp + p - yp) \cdot x$ $\frac{\partial L}{\partial w} = (p-y) \cdot x$ $w_{new} = w_{old} - \eta \cdot (p-y) \cdot x$ $b_{new} = b_{old} - \eta \cdot (p-y)$ (since $\frac{\partial z}{\partial b} = 1$)

in which y and p denote the actual and predicted label of prediction, respectively. w is the weight used in model update that is strongly related to model bias b.

Importance sampling estimator: A local optima γ determines the importance of sampling $\frac{1}{N}\sum_{i=1}^{N} \frac{f(X_i)}{g(X_i)} . f(X_i < \gamma)$ in which g represents sampling density.

Hyperparameter optimizer: Let $\sigma(x)$ is an algorithm that produces an error E. The hyperparameter optimizer H_{opt} aims to minimize E ($H_{opt} \rightarrow min_{\sigma(x)}$) in which x is a set of hyperparameters. Some specific number of subsets $U_1, U_2, ..., U_m$ may constitute x.

LITERATURE REVIEW

Random search was devised to overcome the disadvantages of grid and manual search in large parameter spaces (Bergstra and Bengio 2012). It requires less computational time yet is not suitable for fully controllable experiments, as the number of iterations is set by the practitioner at the beginning. Compared to random search, Bayesian optimization has more potential to achieve sample efficiency for a large number of function evaluations (Turner et al. 2021). Since grid search is time-consuming to evaluate all solution candidates, randomizing the hyperparameter set offers a practical alternative. Hence, a threshold may halt the execution before evaluating the entire grid.

Campbell et al. (2021) developed a Monte Carlo-based hyperparameter optimization method, proposing some updates for the Adam optimizer. They found that Monte Carlo was remarkably helpful for choosing the initial distribution in real-world problems. Large and complex parameter spaces cannot be comprehensively analyzed by exhaustive search algorithms such as grid search. To solve this issue, Baziar et al. (2025) predicted urban water resources using machine learning algorithms, revealing the advantages of Monte Carlo over alternatives. In numerical simulation, Svensson et al. (2015) found that sequential Monte Carlo imposes less computational load compared to Hamiltonian and adaptive importance sampling. In real-world optimization problems, the difference between Monte Carlo and AUTO-SKLEARN Feurer et al. (2022) becomes visible at later stages of tuning (Rakotoarison et al. 2019). Furthermore, although Monte Carlo has the potential to be a good alternative to Bayesian models, it also poses a risk of overfitting. In Jalobeanu et al. (2002), a regularization method was proposed for restoring noisy images. To this end, Monte Carlo was leveraged to predict sampling density. The main limitation of that method is the need for prior knowledge when a unique solution space is desired. When the outputs of a model are very noisy, employing Monte Carlo to predict possible outcomes is a wise choice (Dunbar et al. 2025). However, large-scale hyperparameter transfer methods may be required to know the prior distribution.

Cross-entropy method was originally devised as a minimization algorithm proposed by Rubinstein (1997). It was then used for density estimation tasks in classification problems (Kurian et al. 2021). Cross-entropy is also a feasible algorithm for multi-objective optimization. It yielded promising results for robot path planning (Tang et al. 2024), where various physical constraints were considered during optimization. However, it remains unclear whether the search efficiency depends on the recombination probability of certain adjustment factors. In image processing, cross-entropy strongly depends on the square root error, which leads to an unfavorable convergence rate (Mao et al. 2023). To solve this problem, a generalized bound may be defined for loss minimizers. Cross-entropy may be combined with other probability estimation methods, such as Monte Carlo (Liu et al. 2021). The computational burden stemming from high-dimensional integrals can thus be alleviated. However, its reliability was only tested on numerical functions rather than real-world problems.

METHOD

An overview of the experiment is provided in Figure 1. The process consists of three main steps: data preparation, tuning of MLlib, and results production. It is worth noting that these processes are cyclical, allowing for the creation of a dataset-based configuration pathway to replicate the experiment. RDD conversion is necessary to utilize MLlib algorithms, as they are designed to operate on a

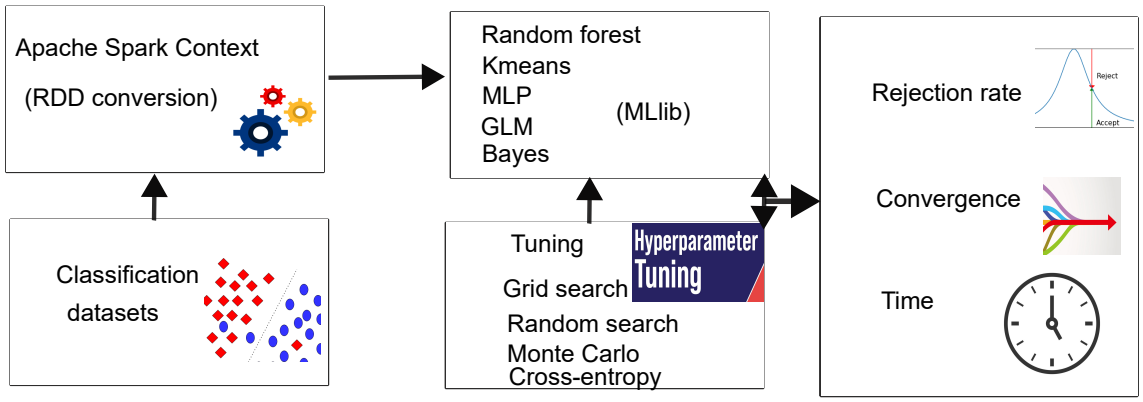

Figure 1 Overview of the experiment.

worker-based structure. Tuning is limited to certain model-based algorithms and two sampling methods. To evaluate performance comprehensively, the experiment was not restricted to a single type of dataset, such as classification. Consequently, the prediction performance of the algorithms was not included in the experimental design.

Algorithm 1 details the use of Monte Carlo methods for optimizing MLlib hyperparameters. Here, *nrep* represents the number of Monte Carlo repetitions; the computational burden of optimization can therefore be adjusted by modifying this parameter. Step 3 establishes a connection to the Apache Spark context. Step 4 converts the dataset into a resilient distributed dataset (RDD). Step 5 splits the dataset into training and testing sets to perform cross-validation. The *result* variable contains the optimal hyperparameters and their corresponding rejection rates.

Algorithm 1 Spark-based MLlib Optimization with Monte Carlo Simulation

1: **Input: nrep, parameters, MLlib algorithm (sparkm), data set (D)**
2: **Output: optimal settings, rejection rates**
3: sc ← spark_connect(master = "local")
4: D_rdd ← copy_to(sc, D)
5: train, test ← sdf_random_split(D_rdd
6: result ← MonteCarlo(sparkm, train, test, parameters)
7: return(result)

In Algorithm 2, *rho* is the elite proportion that denotes the lowest cost of population. *iterThr* is the termination threshold for the optimization iteration. *CEoptim* is a function that is available in R *CEoptim* library. Steps 3-5 are the same with those of Algorithm 1. In Step 7, convergence and ultimate hyperparameter set are returned. R versions of two proposed algorithms can be accessed through [1].

Datasets

Table 1 provides details of the experimental datasets.

The *Dense* dataset was originally created to improve neural network models for classification[2]. Its 30 numerical features can be used to predict a label, making it suitable for binary classification. The *Microsoft* dataset was extracted from a security contest and

[1] https://github.com/muhammedozturk/samplingBasedOptimization
[2] https://www.kaggle.com/c/dense-network/data?select=train.csv

Algorithm 2 Spark-based MLlib Optimization with CE

1: **Input: rho, N, iterThr, parameters, MLlib algorithm (sparkm), data set (D)**
2: **Output: convergence, optimum of function**
3: sc ← spark_connect(master = "local")
4: D_rdd ← copy_to(sc, D)
5: train, test ← sdf_random_split(D_rdd)
6: result ← CEoptim(sparkm, train, test, parameters)
7: return(result)

is publicly available[3]. With 1,804 numerical features (187.83 MB), it can be exploited for various classification experiments. The *Payload* dataset was presented by Politecnico di Milano University and created for a project[4]. Its 31 numerical features can be utilized for classification tasks where data churn is very low (+10, -1). The *Santander* dataset was generated from user transactions[5]. Its 140 numerical features, which consist of floating-point numbers, can be used to design machine learning experiments.

Table 1 MLlib data sets utilized in this work.

Name	Instances	Type
Dense	175000	Classification
Microsoft	10868	Classification
Payload	130529	Classification
Santander	200000	Classification

Experimental settings

Four MLlib algorithms are evaluated in the experiment: Bayes, Kmeans, multi layer perceptron (MLP), and generalized linear model (GLM). Since this study does not consider initialization of

[3] https://www.kaggle.com/muhammad4hmed/malwaremicrosoftbig
[4] https://zenodo.org/record/5731597
[5] https://www.kaggle.com/datasets/lakshmi25npathi/santander-customer-transaction-prediction-dataset

an optimizer, the tuners are run in default configurations. Otherwise, it is another research direction to find the optimal execution settings Monte Carlo and cross-entropy.

Table 2 presents the details of hyperparameters tuned by the comparison algorithms. Note that Bayes has only one hyperparameter (smoothing) that hardens to generalize the result. On the other hand, some hyperparameters such as max_iter and tol are common for GLM, MLP, Kmeans.

Table 2 Details of the hyperparameters used in the experiment.

Algorithm	Hyperparameter	Step size	Range
Random forest	num_trees	1	10-50
Random forest	max_depth	1	2-10
Random forest	subsampling_rate	0.01	0.1-0.9
Bayes	smoothing	1	1-5
GLM	max_iter	1	10-100
GLM	tol	1e-1	1e-09:1e-01
MLP	max_iter	1	10-50
MLP	step_size	0.01	0.1-0.7
MLP	tol	1e-1	1e-09:1e-01
Kmeans	tol	1e-1	1e-09:1e-01
Kmeans	max_iter	1	10-50
Kmeans	k	1	1-5

The machine utilized to perform the experiment has following properties: CentOS Linux, 64-bit, Intel(R) Xenon(R) 2.9 GHz, 24 CPU Cores server with 222 GB RAM. The results given in next section refers to the average values obtained from four datasets.

RESULTS

Figure 2 shows the execution time of the cross-entropy algorithm for 10,000 training instances. Note that each method quickly reaches its peak point before the 200th call. Random Forest requires the most time compared to the others. K-means is the first method that reaches a stagnant line, which occurs at the 140th call. Broadly speaking, it can be concluded that unsupervised methods require less time for cross-entropy optimization. Furthermore, MLP and Bayes have similar fluctuations and cross paths at the 800th call. They can be used interchangeably in Spark for hyperparameter optimization.

The convergence analysis shown in Figure 3 indicates that an rapid drop occurs within a few iterations, regardless of the method used. Optimization time is highly dependent on the algorithm type. In stark contrast, the convergence rate is not affected by the number of iterations and follows nearly the same path throughout the optimization process. It is worth noting that various parameters impact this convergence behavior. First, the experimental data consists primarily of binary classification datasets. Second, MLlib does not allow for the execution of every machine learning algorithm. Instead, it provides a set of well-known algorithms, many of which date back to the early days of machine learning. Consequently, the experiment should be extended to include various deep learning architectures; however, it is currently limited to a feedforward artificial neural network.

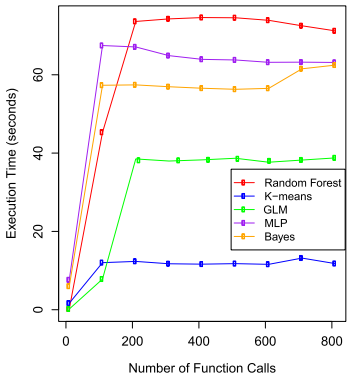

Figure 2 Execution time analysis of MLlib algorithms.

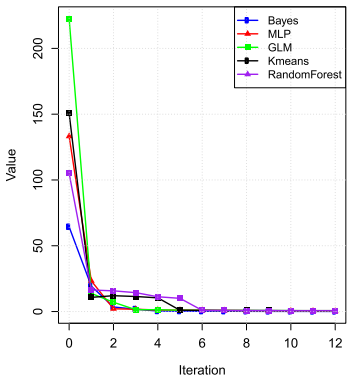

Figure 3 Convergence of MLlib algorithms for cross-entropy optimization.

Figure 4 shows the time competitiveness of the optimizers. Random search is a very stable algorithm, thanks to its threshold settings for iterations. Despite the fact that sampling-based methods provide a critical opportunity to delve into the rationale behind optimal settings beyond mere performance measures, they take much more time than model-free algorithms. In this respect, sampling-based methods should be taken into consideration only when there are restricted computational resources. If an in-depth analysis is not needed to evaluate the hyperparameters, grid search is a good alternative to alleviate the computational burden.

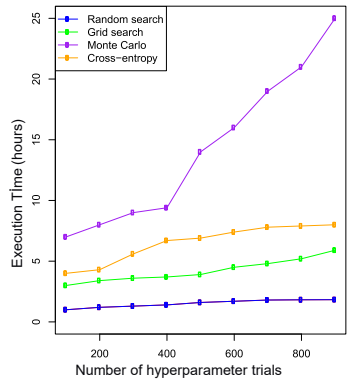

Figure 4 Time comparison of tuners.

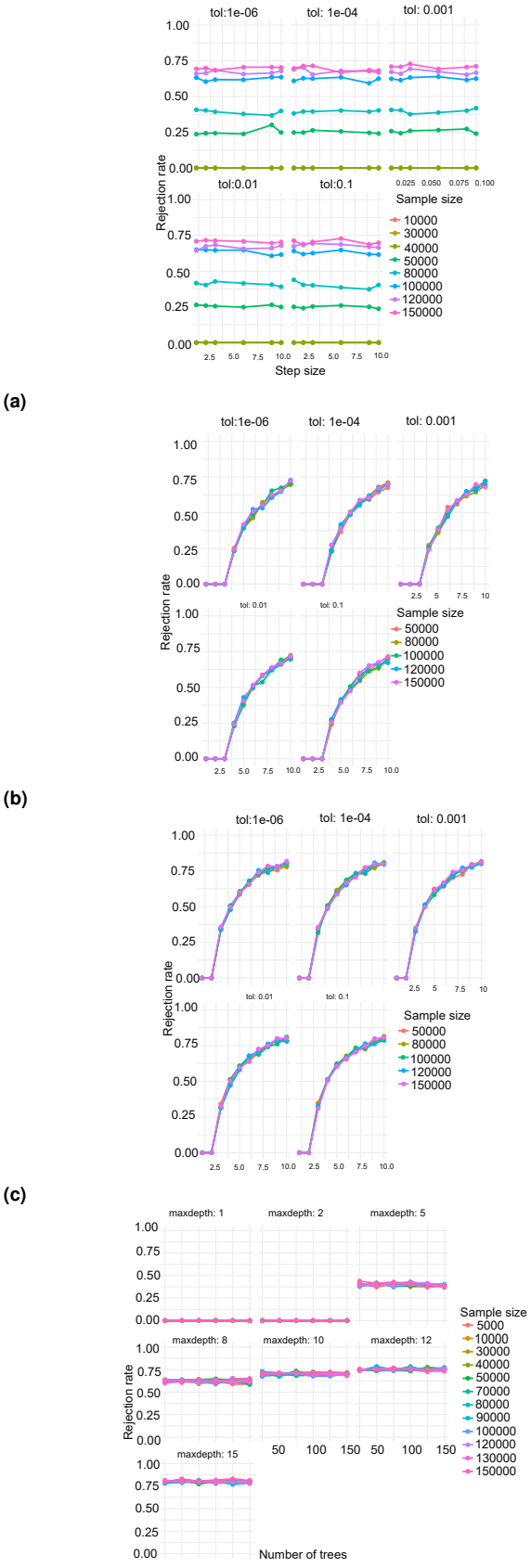

Figure 5 a) MLP b) Kmeans c) GLM d) Random forest

Figure 5 demonstrates the rejection rates of the comparison algorithms for different sample sizes. It is worth noting that a low rejection rate does not necessarily result in a reliable set of optimal hyperparameters. For MLP, a sample size of 10,000 is sufficient to find optimal configurations. The step size is not a critical factor affecting the Monte Carlo process, and the case is similar for the tolerance (tol) parameter. The reasonable number of clusters for K-Means is seven; however, the sample size has a negligible effect on the Monte Carlo performance for this algorithm. A similar pattern was detected for GLM, where 7.5 is the optimal regularization parameter and the number of instances is also not particularly important. For Random Forest, the maximum tree depth should be kept at five to avoid reducing the rejection rate. One can conclude that complex algorithms, such as MLP, are more sensitive to the sampling rate compared to traditional machine learning algorithms in Apache Spark.

Quantitative Analysis of Optimization Challenges in Spark

Table 3 shows the average time per iteration for each optimizer. Monte Carlo and Grid search methods exhibit a 1% and 5% higher average time per iteration compared to Random Search, quantifying the computational overhead introduced by their sampling mechanisms within Spark's RDD-based architecture. On the other hand, Cross-entropy is the second preferable method when considering time-related computational burden. Analysis of Spark event logs revealed that tasks for the Monte Carlo method spent approximately 15% of their runtime on deserialization and data preparation, compared to 5% for Grid Search, indicating the overhead introduced by frequent parameter distribution in sampling-based approaches. While the previous sections analyzed the intrinsic properties of the optimizers, practitioners often operate under strict computational constraints. To provide actionable guidance, we conducted a budget-aware comparison. We imposed three realistic wall-clock time budgets—30, 60, and 120 minutes—on the tuning process for each optimizer and MLlib algorithm. For each run, we recorded the best validation score achieved before the budget was exhausted. The aggregated results are presented in Figure 6.

Table 3 Time analysis based on per iteration.

Algorithm	Avg. Time per Iteration (minutes)	Std. Dev. (minutes)
Monte Carlo	1.4	0.4
Grid search	0.9	0.3
Random search	0.4	0.1
Cross-entropy	0.6	0.2

DISCUSSION

Time complexity is a good indicator for choosing a hyperparameter optimization (HO) method. In this respect, four methods were evaluated based on their time complexities. Grid search has a complexity of $O(N * D)$, where N is the number of candidate values per hyperparameter and D denotes the number of dimensions (hyperparameters). For this reason, grid search is very sensitive to the size of the dimension. The time complexity of random search can be represented as $O(T)$, where T is the number of trials, which limits the total execution time. Monte Carlo has the same complexity, $O(T)$, where T denotes the number of samplings. Note, however,

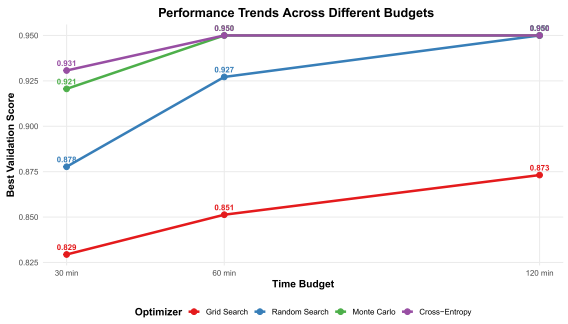

Figure 6 Budget-aware comparison between the optimization techniques .

that Monte Carlo required much more time than random search for tuning due to compatibility issues in Apache Spark. To generalize the results, future experiments could be replicated without using a third-party library to execute Monte Carlo. The highest time complexity belongs to cross-entropy method, at $O(T * M * K)$, where T is the number of iterations, and K and M denote the number of best and elite samples, respectively. If a pre-defined region is used for the optimization, cross-entropy can find an optimal solution at a low cost. Otherwise, it is not very effective for searching the entire solution space.

There are some threats to the validity: 1) The experiment is limited to classification datasets; future work could extend it to other data types, such as regression or clustering datasets. 2) The tuning performance of MLlib was evaluated with various techniques. However, MLlib's native library does not support all machine learning algorithm types. To overcome this issue, its scope should be extended beyond the currently supported feedforward neural network to include other modern architectures.

Our experimental results validate this claim. As shown in Figure 6 and Table 3, Random Search (O(T)) consistently provided the best increase for validation score under the tightest time budgets, making it the most practical choice when computational resources are limited. Conversely, while the Cross-Entropy method has a higher per-iteration cost (O(TMK)), its superior sample efficiency allowed it to achieve the best overall performance when a more generous budget was available. This demonstrates how understanding time complexity guides the practitioner's trade-off between immediate results and ultimate solution quality.

CONCLUSION

Hyperparameter Optimization (HO) is a technique used to enhance the performance of machine learning models. While various HO techniques have been proposed to address specific engineering problems, it remains ambiguous whether sampling-based methods outperform model-free methods. To address this question, this paper presents a comparative analysis between sampling-based and model-free HO approaches. According to the obtained results: 1. Generalized Linear Models (GLM) exhibit fast convergence when combined with cross-entropy optimization. 2. K-means is highly resilient to an increasing number of function calls. 3. Although Monte Carlo sampling provides valuable insights into hyperparameter interactions through its rejection rates, it requires substantial time to complete sampling for each target hyperparameter. The work presented here can be extended through several future research avenues: 1. Derivative-based optimization methods could be compared with sampling-based approaches to develop new HO strategies. 2. The cost-effectiveness of HO could be evaluated on serverless computing platforms. 3. While MLlib currently provides cross-validation for model optimization, Apache Spark should be enriched with native, data-parallel HO libraries instead of relying on third-party solutions to accelerate the tuning process.

Ethical standard

The author has no relevant financial or non-financial interests to disclose.

Availability of data and material

Not applicable.

Conflicts of interest

The author declares that there is no conflict of interest regarding the publication of this paper.

LITERATURE CITED

Andonie, R. (2019). Hyperparameter optimization in learning systems. *Journal of Membrane Computing*, 1(4):279–291.

Assefi, M., Behravesh, E., Liu, G., and Tappert, A. R. (2017). Big data machine learning using apache spark mllib. In *Proceedings of the IEEE International Conference on Big Data*, pages 3492–3498.

Baziar, M., Kashi, A. R., and Karimi, S. M. H. (2025). Machine learning-based monte carlo hyperparameter optimization for thms prediction in urban water distribution networks. *Journal of Water Process Engineering*, 73:107683.

Benham, T., Cole, J. S., and Kloeden, P. E. (2017). Ceoptim: Cross-entropy r package for optimization. *Journal of Statistical Software*, 76:1–29.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(1):281–305.

Beruvides, G., Quiza, R., and Castaño, F. (2016). Multi-objective optimization based on an improved cross-entropy method: A case study of a micro-scale manufacturing process. *Information Sciences*, 334:161–173.

Campbell, A., Chen, J., and Li, W. (2021). A gradient based strategy for hamiltonian monte carlo hyperparameter optimization. In *Proceedings of the International Conference on Machine Learning*, pages 1238–1248.

Cao, R., Chen, Y., and Zhang, W. (2024). Etune: Efficient configuration tuning for big-data software systems via configuration space reduction. *Journal of Systems and Software*, 209:111936.

Dunbar, O., Garcia-Trillos, N., and Perego, M. (2025). Hyperparameter optimization for randomized algorithms: a case study on random features. *Statistics and Computing*, 35(3):1–28.

Eleftheriadis, P., Lioudakis, G., and Amditis, A. (2024). Joint state of charge and state of health estimation using bidirectional lstm and bayesian hyperparameter optimization. *IEEE Access*, 12:80244–80254.

Feurer, M., Eggensperger, K., Falkner, S., Lindauer, M., and Hutter, F. (2022). Auto-sklearn 2.0: Hands-free automl via meta-learning. *Journal of Machine Learning Research*, 23(261):1–61.

Feurer, M. and Hutter, F. (2019). Hyperparameter optimization. In *Automated Machine Learning: Methods, Systems, Challenges*, pages 3–33. Springer, Cham, Switzerland.

Herbst, P., Schuld, M., and Petruccione, F. (2024). On optimizing hyperparameters for quantum neural networks. In *Proceedings of the IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 1, pages 1478–1489.

Ilmeboya, J., Adebayo, P. O., and Adewumi, A. O. (2024). Hyperparameter tuning in machine learning: A comprehensive review. *Journal of Engineering Research and Reports*, 26(6):388–395.

Jalobeanu, A., Zerubia, J., and Blanc-Talon, J. (2002). Hyperparameter estimation for satellite image restoration using a mcmc maximum-likelihood method. *Pattern Recognition*, 35(2):341–352.

Khaldi, M., Kafi, A. E., and Bernoussi, S. E. (2025). Hyperparameter optimization for malicious url detection: Leveraging optuna and random search in machine learning and deep learning models. *Informatica*, 49(27).

Kurian, N. C., Dubey, S. R., and Chakraborty, S. (2021). Sample specific generalized cross entropy for robust histology image classification. In *Proceedings of the IEEE 18th International Symposium on Biomedical Imaging*, pages 1934–1938.

Li, G., Zhao, W., and Wang, Y. (2019). An improved butterfly optimization algorithm for engineering design problems using the cross-entropy method. *Symmetry*, 11(8):1049.

Liu, X., Zhang, Y., and Chen, W. (2021). A cross-entropy algorithm based on quasi-monte carlo estimation and its application in hull form optimization. *International Journal of Naval Architecture and Ocean Engineering*, 13:115–125.

Mao, A., Mohri, M., and Abu-Mostafa, Y. S. (2023). Cross-entropy loss functions: Theoretical analysis and applications. In *Proceedings of the International Conference on Machine Learning*, pages 23803–23828.

Meister, M., Parnell, T., and Zhang, C. (2020). Maggy: Scalable asynchronous parallel hyperparameter search. In *Proceedings of the 1st Workshop on Distributed Machine Learning*, pages 28–33.

Meng, X., Bradley, J. K., Yavuz, B., Sparks, E. R., Venkataraman, S., and Franklin, M. J. (2016). Mllib: Machine learning in apache spark. *Journal of Machine Learning Research*, 17(34):1–7.

Naeem, M., Reza, S. M. S., and Kemp, A. H. (2009). Cross-entropy optimization for sensor selection problems. In *Proceedings of the 9th International Symposium on Communications and Information Technology*, pages 396–401.

Nguyen, N., Hassan, M. A., Bai, K., and Wang, Y. (2018). Towards automatic tuning of apache spark configuration. In *Proceedings of the IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 417–425.

Rakotoarison, H., Sebag, M., and Schoenauer, M. (2019). Automated machine learning with monte-carlo tree search. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 3296–3303.

Rubinstein, R. Y. (1997). Optimization of computer simulation models with rare events. *European Journal of Operational Research*, 99(1):89–112.

Svensson, A., Schön, T. B., and Kok, M. (2015). Marginalizing gaussian process hyperparameters using sequential monte carlo. In *Proceedings of the IEEE 6th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, pages 477–480.

Tang, Q., Wang, Y., and Liu, Z. (2024). A dual-robot cooperative arc welding path planning algorithm based on multi-objective cross-entropy optimization. *Robotics and Computer-Integrated Manufacturing*, 89:102760.

Turner, R., Eriksson, D., McCourt, M., Kiili, J., Laaksonen, E., and Xu, Z. (2021). Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020. In *NeurIPS 2020 Competition Demonstration Track*, pages 3–26.

van Stein, N., Bäck, T., and Emmerich, M. (2024). In-the-loop hyper-parameter optimization for llm-based automated design of heuristics. *ACM Transactions on Evolutionary Learning*.

Zhou, M., Li, Y., and Chen, W. (2025). Towards hybrid architectures for big data analytics: Insights from spark-mpi integration. *IEEE Transactions on Services Computing*, pages 1852–1868.

Zhu, Y., Wang, X., and Zhang, L. (2025). Rockhopper: A robust optimizer for spark configuration tuning in production environment. In *Companion Proceedings of the International Conference on Management of Data*, pages 743–756.

How to cite this article: Öztürk, M. M. Hyperparameter Optimization for Big Data: Adapting Sampling Methods to Apache Spark MLlib. *ADBA Computer Science*, 3(1), 6-12, 2026.