A Non-Deterministic Strategy for Searching Optimal Number of Trees Hyperparameter in Random Forest

Kennedy Senagi
Department of Information Technology
Dedan Kimathi University of Technology
Kenya
Email: kennedy.senagi@dkut.ac.ke

Nicolas Jouandeau
LIASD
University Paris8
France
Email: n@ai.univ-paris8.fr

***Abstract*—In this paper, we present a non-deterministic strategy for searching for optimal number of trees hyperparameter in Random Forest (RF). Hyperparameter tuning in Machine Learning (ML) algorithms is essential. It optimizes predictability of an ML algorithm and/or improves computer resources utilization. However, hyperparameter tuning is a complex optimization task and time consuming. We set up experiments with the goal of maximizing predictability, minimizing number of trees and minimizing time of execution. Compared to the deterministic search algorithm, the non-deterministic search algorithm recorded an average percentage accuracy of approximately 98%, number of trees percentage average improvement of 44.64%, average time of execution mean improvement ratio of 213.25 and an average improvement of 94% iterations. Moreover, evaluations using Jackkife Estimation show stable and reliable results from several experiment runs of the non-deterministic strategy. The non-deterministic approach in searching hyperparameter shows a significant accuracy and better computer resources (i.e cpu and memory time) utilization. This approach can be adopted widely in hyperparameter tuning, and in conserving utilization of computer resources like green computing.**

I. Introduction

ML performance tuning is aimed at improving the predictability of ML algorithms. Improving performance of a ML systems can be done by configuring a set of hyperparameters. Most ML algorithms have several hyperparameters to be configured. Hyperparameters specify the interoperability of the underlying model. ML algorithms hyperparameter tuning is aimed at getting optimal values that can improve the algorithm's predictability considering minimum consumption of computer system resources [6]. When adopting ML algorithm to a specific dataset, hyperparameter tuning can be cumbersome and time consuming [13].

Manual, grid search and bayesian optimization are methods of hyperparameter optimization. Grid search is deterministic. It does an exhaustive search. It uses a predefined parameter space $S = \{0, 1, 2, ..., n\}$. The goal is to search an optimal hyperparameter s in S that records an optimal accuracy. Grid search consumes substantial amount time and is computationally expensive. However, it gives accurate results [4]. Manual search involves randomly selecting a value s in S. The value s is configured in the algorithm, the experiment executed and the accuracy observed. The process is repeated comparing the accuracy. The hyperparameter that records the optimal accuracy is selected. Manual search is cumbersome and difficult to reproduce results [1]. Bayesian optimization stochastically and efficiently trades off exploration and exploitation of the parameter space. It also explores historical information to find the parameters that maximize functions to inform user the configurations that best optimize predictability of the ML algorithm [5].

This paper introduces a non-deterministic search algorithm. The algorithm randomly selects 10% of elements in a parameter space. It then uses heuristics and termination conditions to maximize accuracy (acc) and minimize time of execution (t). This algorithm was applied and tested in selecting optimal number of trees (θ) in random forest (RF). In this paper, Section II covers related works, Section III discusses methodology and Section IV concludes this paper.

II. Related Works

In the paper by Hazan et al. (2017), large scale machine learning systems at times involves large number of parameters that are fixed manually. This is time consuming and at times inaccurate and difficult for a human expert. A hyper-parameter optimization strategy is proposed inspired by analysis of boolean function focusing on high-dimension datasets. The algorithm is an iterative application of compressed sensing techniques for orthogonal polynomials. The algorithm is tested in deep neural networks. In terms of running time, the algorithm records at least an order of magnitude faster than Hyperband and Bayesian Optimization and outperform Random Search 8x [Hazan et al., 2017]. Hazan et al. (2017) guides this work as they develops an algorithm and tests it in another algorithm; their algorithm establishes heuristics for reducing the search space.

Experiments showed that accuracy increased when number of trees in RF was doubled. However, there was a threshold beyond which there was no significance gain in accuracy. Therefore, increasing number of trees does not always mean

a better performance can be attained [15]. We note that, there was no significant variable that used to measure use of computing resources consumed when varying number of trees.

MapReduce was used to optimize regularization parameters for boosted trees and random forests (RF). For RF[2], two parameters were tuned: the number of trees in the model and the number of features selected to split each node. Experiments showed that performance was sensitive to the number of trees but less sensitive to the number of features in each split. Results showed that MapReduce could make parameter optimization feasible on a massive scale. However, it created possibilities for overfitting that could reduce accuracy and lead to inferior learning parameters [6].

In the technical report by [3], they discuss manually setting up, using and understanding RF. They note that RF grows trees rapidly and setting up a large number of trees (e.g. 1000) is okay. They further note that, if there are many variables, they can grow more trees (of up-to 5000) Beiman, (2003). From this work we can set up experiments with variable number of trees and see their effects on computing resources.

ML algorithms often involve careful tuning of learning parameters and model hyper-parameters. Parameter tuning is often a "black art" that requires expert experience, rules of thumb or sometimes brute-force search. To solve this problem, the following techniques were used: a full Bayesian treatment expected improvement, and algorithms (e.g ANN) for dealing with variable time regimes and running experiments in parallel. Results of this experiment surpassed a human expert at selecting hyper-parameters on the competitive CIFAR-10 dataset; beating the state of the art by over 3%. SVM was used as a case study algorithm [13].

A novel idea for approximate tree learning is seen in sparsity-aware algorithm for sparse data and weighted quantile sketch. The algorithm (XGBoost) proposes candidate splitting points according to percentiles of feature distribution, then maps the continuous features into buckets split, aggregates the statistics and finds the best solution among proposals based on the aggregated statistics. The algorithm also provides an insights on cache access patterns, data compression and sharing to build a scalable tree boosting system. The algorithm has been widely used and recognized in machine learning and data mining challenges e.g. Kaggle and KDDCup 2015. The algorithm can be applied to machine learning systems and in solving real-world scale problems using a minimal amount of resources [4].

Optimizing parameters of an evolutionary algorithm values is a challenging activity. CMA-ES tuning algorithms gave better results in terms of utility, in evolution algorithms. It is noted that using algorithms for tuning parameters of evolutionary algorithms does pay off in terms of performance. However, tuning algorithms gave better tuning parameter values than relying on intuitions and the usual parameter setting conventions [14].

It is challenging to create a large dataset and improve train ability of deep neural network models (DNNs). A selection of supplemental training datasets was used in fine-tuning a high-performing neural network model. Natural Language Processing system ability is improved after being evaluated by the Item Response Theory ability scores without negatively affecting generalization due to overfitting [9].

Large scale machine learning systems at times involve large number of parameters that are fixed manually. This is time consuming and at times inaccurate and difficult for a human expert. A hyper-parameter optimization strategy is proposed inspired by analysis of boolean function focusing on high-dimension datasets. The algorithm is an iterative application of compressed sensing techniques for orthogonal polynomials. The algorithm is tested in deep neural networks. In terms of running time, the algorithm records at least an order of magnitude faster than Hyperband and Bayesian Optimization and outperform Random Search 8x. The algorithm requires only uniform sampling of the hyperparameters and is easily parallelizable [7].

In the department of Soil Survey in Kenya Agriculture and Livestock Research Organization (KALRO) [10] and other soil research organizations, land evaluation is done manually, is stressful, takes a long time and is prone to human errors [11][12]. Parallel RF experiment prototypes are set up in [11] and further experiments in [12]. Parallel RF, Linear Regression, Linear Discriminant Analysis, KNN, Gaussian Naive Bayesian and Support Vector Machine are applied in predicting land suitability for crop (sorghum) production, given soil properties information. Parallel RF had a better accuracy of 0.96 and time of execution of 1.7 sec [12].

Besides assertions regarding performance reliability of default parameters in RF, many RF experiments fit using these values. An examination of parameter sensitivity of RF in computational genomic was studied. Experiments were evaluated using Area Under Curve (AUC), Root Mean Square Error (RMSE) and cross-fold validation. It was seen that RF performance was strongly affected by number of trees, sample size and number of random variables used at each split. It was noted that tuned RF gave better results than when default parameters/values are used. Effects of parameterization were analyzed using selection methods and showed that tuning can successfully improved prediction accuracy of non-parametric ML algorithms [8].

III. Methodology

In this research, we considered 14 standardized datasets collected from UCI Machine Learning website, namely: Balance Scale (1), Breast Cancer Wisconsin - Original (2), Car Evaluation (3), Habermans Survival (4), Pen-Based Recognition of Handwritten Digits (5), Website Phishing (6), Yeast (7), Banknote Authentication (8), Contraceptive Method Choice (9), Diabetic Retinopathy Debrecen (10), EEG Eye State (11), Pima Indians Diabetes (12), Wine Quality - White (13) and Wine Quality (14). In each dataset, we used simple random sampling without replacement strategy to sample 10% of elements in the search space. All experiments were run 10 times and results averaged. Number of trees (θ) was varied accordingly as we measured accuracy (acc) and time of

execution (t). The computer had the following specifications: Intel(R) Xeon(R) CPU W3505 @ 2.53GHz x 2.

A. Considering 2 to 4096 Number of Trees

We considered a finite set of sorted number of trees in the parameter space. RF predictability was evaluated by acc defined in equation 1 with n samples, where $\hat{y}_i$ is the predicted label and y_i is the original label. The results of acc and t are tabulated in Tables I and II respectively.

$$acc(y, \hat{y}) = \frac{1}{n} \sum_{i=0}^{n-1} 1(\hat{y}_i = y_i) \tag{1}$$

Table I shows a general trend of accuracy increasing steadily with increase in number of trees, then flattens. RF classification employs bagging principles, where a committee of trees each, cast a vote for the predicted class. However, RF classifier introduces modifications in bagging where it builds a large collection of de-correlated trees, and then averages them. When the number of trees become huge, we see RF accuracy varying insignificantly meaning the average accuracy of de-correlated trees varying insignificantly. Average accuracy varies because of the random nature of RF, for example, randomly selecting features when building trees. We further observed an interesting trend in the number of trees against accuracy; increasing the number of trees does not significantly contribute to a positive accuracy. The maximum accuracy values are in bold, in Table I. Moreover, we see 13 out of the 14 dataset's maximum accuracy values found between 2 and 512 trees. Dataset 6 with 2048 number of trees recorded an accuracy of 88.7% and 6.42 seconds. It's second best accuracy is 88.4% with 0.89 seconds observed at 256 number of trees. In this case, we think 256 number of trees is better because the change in accuracy rather insignificant (-0.3) while it runs faster (approximately 7x faster). Generally, we observed better results between 2 and 512, and we assume these results can be extended to other datasets. We call the region between 2 and 512, the $fertile\ region$.

Table II shows a general trend of time of execution increasing steadily with increase in number of trees. This tells us that more number of trees demand more computing resources. We also observed a relative significant change in time of execution, the threshold values are in bold. Generally, after 64 number of trees, we see a significant change in time difference. Increase in number of trees increases time of execution. More number of trees requires more computer resources to build and average the de-correlated trees in RF.

Different datasets give different values of accuracy and time of execution with the same number of trees. The selected datasets have different complexity i.e dimensionality, number of records and classes. This leads to a variation in accuracy and time of execution. For us to have an optimal number of trees hyperparameter in RF classifier, it is important we consider maximizing accuracy and minimizing number of trees.

However, we see the 6$^{\text{th}}$ dataset maximum accuracy of 88.7% and time of execution of 6.42 seconds being out of the fertile region i.e 2048 number of trees. As per our experiments, this is a probability of 0.07 i.e 1 out of 14 datasets can exhibit this. The second best accuracy of 87.9% is observed in the fertile region i.e 128 number of trees with 0.5 seconds time of execution. In such instances, we can compromise accuracy to get a better time of execution, for this case, we compromise 0.8% accuracy to gain 5.92 seconds.

B. Considering 2 to 512 Number of Trees

In the fertile region, we observed lower time of execution and maximum accuracy, therefore, we will have avoided searching out regions (> 512) that show higher time of execution and significantly same or lower accuracy. We defined a finite set of sorted number of trees from the parameter space θ. We configured, trained and tested RF with the respective θ and recorded acc and t. The results are show in Fig. 1 and 2. Fig. 1 is a box plot of accuracy for number of trees against datasets across 14 datasets in the fertile region. Most datasets had a low inter-quartile range, low difference between the low and maximum points and more outliers below the lower whiskers. Some box plots also recorded some outliers above the upper whisker. A low difference in quartile ranges means there was a low variation in accuracy from the median and 50% of the accuracy records are within this region. However, the outliers inform us that, some maximum accuracy values were very far away from the median and some lowest accuracy values were very far away from the median. The goal of any data scientist is to have the maximum accuracy when configuring RF with a specific number of trees. Nonetheless, we see variations in accuracy on different datasets, i.e. different datasets record different accuracy levels. This make the search problem more difficult because we need to have a strategy that will be dynamic to search the best accuracy in different datasets. This research was interesting in finding number of trees (i.e. the outliers in the upper whisker) that maximize accuracy.

Fig. 2 is a box plot of time of execution of number of trees against datasets across 14 datasets in the fertile region. We see the lower whisker having almost the same time of execution. This means there are some number of trees that could give almost the same minimum time of execution when configured in RF. We also see the lower whiskers being shorter than the upper whiskers. A shorter lower whisker means most lower time of executions were closer to the median. This research was interesting in these number of trees that minimize time of execution.

From these analysis, we formulated deterministic, non-deterministic and automatic configuration (having 8 number of trees by default) algorithmic approaches in searching optimal number of trees hyperparameter in the fertile region.

C. Deterministic Hyperparameter Search

Deterministic search algorithm is defined in equation 2. We developed a deterministic hyperparameter search algorithm from equation 2 as outlined in Algorithm 1. We considered number of trees θ, time t and accuracy acc descriptions and results from Section III-B. The deterministic hyperparameter search algorithm's goal is to maximize acc and minimize θ.

Table I: Accuracy (percentage) of RF with θ trees for 14 datasets (DS)

DS	Number of Trees											
	2	4	8	16	32	64	128	256	512	1024	2048	4096
1	80.3	81.9	83.0	82.4	84.6	**85.6**	84.6	84.0	84.0	84.0	84.6	84.6
2	91.7	93.7	97.1	**98.0**	97.6	97.6	97.6	97.1	97.1	97.1	97.1	97.1
3	86.3	85.5	83.6	83.8	**84.8**	84.4	84.6	84.4	84.8	84.8	84.6	84.6
4	76.1	79.3	75.0	76.1	**79.3**	79.3	78.3	78.3	78.3	79.3	78.3	79.3
5	92.5	96.8	98.3	98.6	98.4	98.9	99.0	**99.1**	99.0	99.1	99.1	99.1
6	81.5	86.9	86.2	87.4	85.7	87.4	87.9	88.4	87.7	87.9	**88.7**	88.2
7	48.6	47.8	52.9	57.3	56.5	59.5	**59.8**	58.8	58.8	58.5	58.5	58.8
8	96.6	**97.8**	97.6	97.6	97.3	97.6	97.8	97.8	98.1	97.8	97.8	97.8
9	46.4	48.4	49.1	**51.6**	49.5	49.8	51.1	49.5	50.7	51.4	50.9	51.1
10	61.3	64.7	65.3	65.0	**69.9**	66.5	67.6	67.9	68.2	67.1	67.9	67.3
11	77.9	84.2	87.9	89.3	91.3	**92.7**	92.0	92.2	92.2	92.1	92.3	92.2
12	66.7	71.0	74.9	74.5	76.6	76.6	76.6	75.8	**77.5**	76.6	77.1	77.1
13	54.9	59.4	64.7	64.6	65.7	65.9	67.1	**67.3**	67.1	66.6	67.3	67.4
14	54.4	69.7	63.3	67.3	69.2	69.2	69.6	**70.2**	69.8	69.2	69.8	69.8

Table II: Time of execution (sec) of RF with θ trees for 14 datasets (DS)

DS	Number of Trees											
	2	4	8	16	32	64	128	256	512	1024	2048	4096
1	0.21	0.21	0.22	0.23	0.25	**0.30**	0.51	0.90	1.60	3.29	6.49	12.45
2	0.21	0.21	0.22	0.23	0.25	**0.30**	0.50	0.90	1.59	3.09	5.98	12.35
3	0.21	0.21	0.22	0.23	0.25	**0.30**	0.50	1.00	1.80	3.39	6.79	13.57
4	0.21	0.21	0.22	0.23	0.25	**0.30**	0.50	0.80	1.60	3.30	5.99	12.06
5	0.21	0.21	0.22	0.23	**0.26**	0.41	0.60	1.10	2.20	4.01	8.23	15.87
6	0.21	0.21	0.22	0.23	0.25	**0.30**	0.50	0.89	1.89	3.46	6.42	13.14
7	0.21	0.21	0.22	0.23	0.26	**0.30**	0.50	1.00	1.88	3.71	7.13	14.06
8	0.21	0.21	0.22	0.23	0.25	**0.30**	0.50	0.90	1.69	3.17	6.64	12.76
9	0.21	0.21	0.22	0.23	0.25	**0.30**	0.50	1.00	1.89	3.47	6.95	13.70
10	0.21	0.21	0.22	0.23	0.25	**0.30**	0.50	0.90	1.79	3.47	6.93	14.41
11	0.21	0.21	0.22	0.33	**0.46**	0.71	1.20	2.40	4.70	9.18	18.27	36.62
12	0.21	0.21	0.22	0.24	0.25	**0.30**	0.50	0.79	1.68	3.46	6.43	12.64
13	0.21	0.21	0.22	0.23	**0.25**	0.51	0.70	1.40	2.69	5.28	10.45	21.20
14	0.21	0.21	0.22	0.23	0.25	0.40	**0.60**	1.10	2.09	3.76	7.33	14.96

We note that, $\exists acc_{max} \in acc$ that has θ_{best}. The deterministic search algorithm is exhaustive, i.e., it does a linear search and returns acc_{max}, with θ_{best} and the time needed t. Experiment results are tabulated in Tables III, IV and V.

$$\theta^*_{best}, acc^*_{best} = \underset{\theta \in \mathcal{T}}{argmax}\ \hat{Q}\ (\theta, acc) \tag{2}$$

Algorithm 1 The Deterministic Hyperparameter Search

```
procedure DETERMINISTICSEARCH(train, test)
    t_i ← CURRENTTIME()
    T ← [θ_1, θ_2, θ_3, ..., θ_n]
    acc_max ← 0
    for each θ in T do
        rf ← RANDOMFOREST(θ, train)
        acc_new ← ACCURACY(rf, test)
        if acc_new > acc_max then
            (acc_max, θ_best) ← (acc_new, θ)
    time_spent ← CURRENTTIME() − t_i
    return (acc_max, θ_best, time_spent)
```

D. The Non-Deterministic Hyperparameter Search Algorithm

In this research, we were interested in maximizing accuracy and minimizing number of trees. Tables 1 and 2 shows almost the same accuracy but with different time of execution. Table 2 shows more NoTs require more ToE (i.e. memory and cpu time). With this analogy, this research formulated a non-deterministic search approach to converge close/to maximize accuracy and minimize number of trees and save time of execution. The algorith is outlined Algorithm 2, where $\theta_i = random(\in \mathcal{T}), \psi_1 = 1 + \frac{lim}{100},\ and\ \psi_2 = 1 - \frac{lim}{100}$.

We considered θ, acc and t descriptions and results from Section III-B. The goal of this algorithm was to maximize acc and minimize t through randomization. In this algorithm we assumption that, $\exists acc_{best} \in acc$ that has θ_{best}. Note that the function GENERATE( ) returns 26 elements which is approximately 10% of elements in the parameter space. We iterate through the random selected number of trees as we configure RF. We considered percentage upper bound and lower bound of the acc_{best}. If acc_{rand} falls in the upper boundary, then $acc_{best} \leftarrow acc_{rand}, \theta_{best} \leftarrow \theta_{rand}$ and we $break$, with the assumption that we do not anticipate further percentage Δ acc_{best}. If acc_{rand} falls in the lower boundary and θ_{rand} is less than θ_{best}, then $acc_{best} \leftarrow acc_{rand}, \theta_{best} \leftarrow \theta_{rand}$ and we also $break$, with the assumption that we have an insignificant Δ acc_{best} and we have a better t_{best}. Moreover, if acc_{rand} falls above the upper boundary, then $acc_{best} \leftarrow acc_{rand}, \theta_{best} \leftarrow \theta_{rand}$, and we continue looping with the assumption that

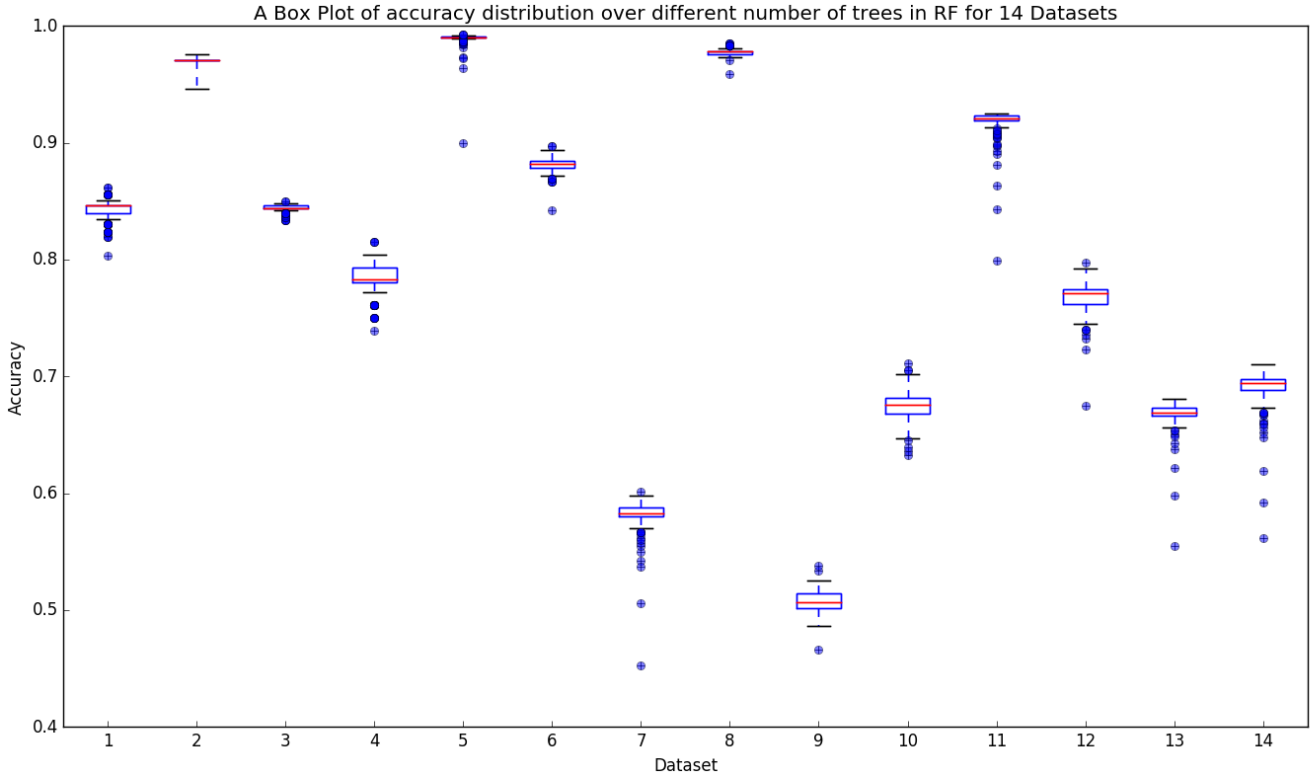

Fig. 1. Number of trees (many) against datasets of Accuracy in RF for 14 Datasets

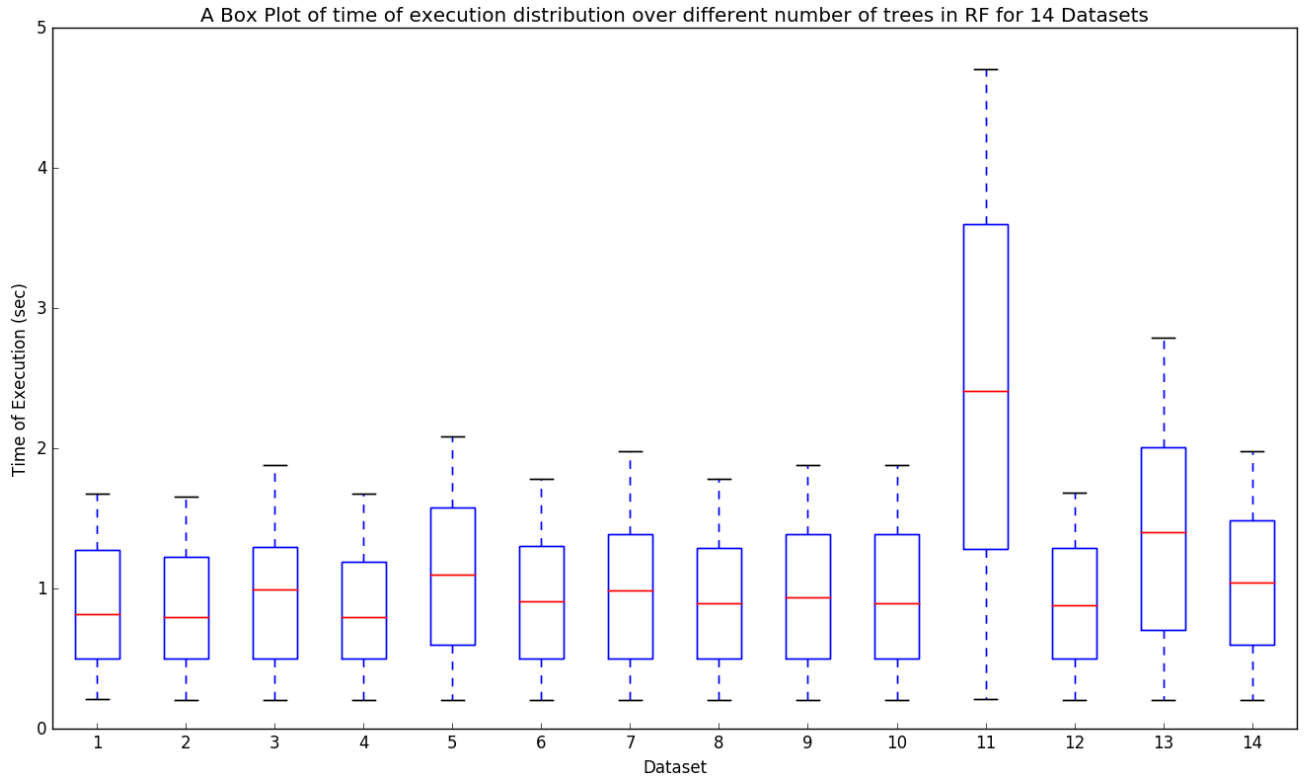

Fig. 2. Number of trees (many) against datasets of Time of Execution in RF for 14 Datasets

we anticipate further percentage Δ acc_{best}. Lastly, we $break$ when iteration counts are 10% of the parameter space, with the assumption that we have uniformly sampled the whole parameter space. We set the percentage boundary as 1% to increase the algorithm's accuracy. Experiment results are tabulated in Tables III, IV and V.

Algorithm 2 The Non Deterministic Hyperparameter Search

$vals = []$
procedure GENERATE()
  while LEN($vals$) ≤ 26 **do**
    $val = 2 + rand()\%512$
    if val *is not in* $vals$ **then**
      add val *in* $vals$
  return val
procedure NONDETERMINISTICSEARCH($train, test$)
  $t_i \leftarrow$ CURRENTTIME()
  $acc_{rand}, \theta_{rand}, acc_{best}, \theta_{best}, count \leftarrow 0$
  $\mathcal{T} \leftarrow$ GENERATE( )
  for each θ_{rand} **in** $\mathcal{T}$ **do**
    $rf \leftarrow$ RANDOMFOREST($\theta_{rand}, train$)
    $acc_{rand} \leftarrow$ ACCURACY($rf, test$)
    if $count == 0$ **then**
      $(acc_{best}, \theta_{best}) \leftarrow (acc_{rand}, \theta_{rand})$
    if $\psi_1.acc_{best} > acc_{rand} > \psi_2.acc_{best}$ **then**
      if $acc_{rand} < acc_{best}$ **then**
        if $\theta_{rand} < \theta_{best}$ **then**
          $(acc_{best}, \theta_{best}) \leftarrow (acc_{rand}, \theta_{rand})$
          break
      else
        $(acc_{best}, \theta_{best}) \leftarrow (acc_{rand}, \theta_{rand})$
        break
    else if $acc_{rand} > \psi_1.acc_{best}$ **then**
      $(acc_{best}, \theta_{best}, count) \leftarrow (acc_{rand}, \theta_{rand}, 0)$
    $count \leftarrow count + 1$
    if $count >= 10$ **then break**
  $time\_spent \leftarrow$ CURRENTTIME() $- t_i$
  return $(acc_{best}, \theta_{best}, time\_spent)$

E. Determinstic and Non-Deterministic Hyperparameter Search Algorithms, and Auto-Configured RF

Table III contains results and analysis of minimum number of trees selected by deterministic and non-deterministic hyperparameter search algorithms. We see a considerably good percentage improvement of number of trees in the non-deterministic search algorithm. At some instances, for example, in datasets 8 and 13, the non-deterministic search algorithm was able to perfectly converged to the minimum number of trees with 26 and 2 iterations respectively. In some datasets e.g dataset 1, the percentage number of trees improvement was poor. Moreover, as observed in Table III, 50% of the datasets used less than 50% (i.e. less than 5% of random values in the search space) of random values while iterating, to converge close/to maximum accuracy and minimum number of trees. With this observation, in some cases, we can have an assumption that sometimes increasing the search space would not have much scientific significance. Generally, the percentage number of trees improvement was 44.6% and the average number of iterations used were 14.5.

Table IV has results and analysis of accuracy recorded from running deterministic, non-deterministic and auto-configured RF algorithms. The auto-configured RF had a mean percentage difference -4.9 while the non-deterministic search algorithm had a considerably better percentage change of -1.69. In non-deterministic search algorithm, datasets 2, 8 and 13 recorded a zero percentage change in accuracy. 50% of the datasets recorded a percentage change of more than 1%.

Table V has results and analysis of time of execution of deterministic and non-deterministic search algorithms, and auto-configured RF. The ratio of deterministic:non-deterministic algorithms and deterministic: auto-configured RF are calculated. Their averages are also calculated. Both auto-configured RF and non-deterministic algorithm record a very high average ratio of 6827 and 213 respectively.

As discussed in Section III-C, the deterministic search algorithm is exhaustive and selects the minimum number of trees that has the maximum accuracy. With these results, we benchmark the non-deterministic search algorithm and auto-configured RF. The non-deterministic search algorithm, as discussed in Section III-D, uses the principle of randomization, heuristics and terminating policies as outlined in Algorithm 2. With this strategy, the non-deterministic search algorithm recorded $\approx$ 98% average accuracy, and could run at an average of 212.79 faster, on an average of 14.5 iterations. Using the strategy formulated in Algorithm 2, the non-deterministic search algorithm recorded 100% accuracy at three instances and recorded zero number of trees percentage improvement on two instances. Moreover, in the non-deterministic search algorithm, we recorded number of trees that are below the number of trees threshold (64 trees), that showed a significant change in time of execution, as discussed in Section III-A. This means the formulated strategy worked quite well. Considering dataset 2, we note that 0% percentage accuracy change, was got with more number of trees (48 trees instead of 46 trees) but at 34.76 times faster. These shows 100% accuracies got, at more number trees but takes a shorter searching time. This makes the strategy formulated in this research more relevant. Despite the 1% boundary policy and breaking policies strategies, 50% of the datasets recorded less than 1% change in percentage accuracy. The other 50% scored fairly good results too. Generally, a shorter time of execution means the process will take a shorter time in memory and shorter cpu time, when tuning RF. We see the non-deterministic search algorithm run $\approx$ 213 faster on average, achieving an average of $\approx$ 98% accuracy, on an average of 5.6% iterations (i.e 14.5 of 256 iterations in the parameter space). This is an improvement in iterations by 94.4%. Therefore, the non-deterministic search algorithm can improve utilization of computing resources while maintaining a significant accuracy.

Auto-configuring (having 8 number of trees by default) RF

Table III: Recorded minimum number of trees (θ_{best}) and iterations for deterministic and non-deterministic search algorithms across 14 datasets (DS), and their mean (μ)

DS	Deterministic	Non-Deterministic		
	θ_{best}	θ_{best}	θ % improvement	Iteration
1	26	32	-23.08	5
2	46	48	-4.35	26
3	116	46	60.34	26
4	70	18	74.29	26
5	48	16	66.67	26
6	216	26	87.96	26
7	118	34	71.19	3
8	44	44	0.00	26
9	48	42	12.50	2
10	18	10	44.44	4
11	196	50	74.49	26
12	164	10	93.90	2
13	46	46	0.00	2
14	150	50	66.67	3
μ	93.28	33.71	44.64	14.5

Table IV: Maximum accuracy (acc_{best}) recorded across 14 datasets (DS), and their mean (μ)

DS	Deterministic	Auto-Configured		Non-Deterministic	
	acc_{max}	acc_{best}	% Δ	acc_{best}	% Δ
1	0.862	0.819	-4.99	0.856	-0.70
2	0.976	0.971	-0.51	0.976	0.00
3	0.850	0.846	-0.47	0.846	-0.47
4	0.815	0.761	-6.63	0.804	-1.35
5	0.993	0.973	-2.01	0.990	-0.30
6	0.897	0.855	-4.68	0.887	-1.11
7	0.601	0.552	-8.15	0.593	-1.33
8	0.985	0.976	-0.91	0.985	0.00
9	0.538	0.480	-10.78	0.505	-6.13
10	0.711	0.627	-11.81	0.682	-4.08
11	0.925	0.890	-3.78	0.919	-0.65
12	0.797	0.740	-7.15	0.736	-7.65
13	0.681	0.636	-6.61	0.681	0.00
14	0.710	0.654	-7.89	0.679	-4.37
μ	0.76	0.72	-4.9	0.747	-1.7

Table V: Time of execution (sec) recorded across 14 datasets (DS), and their mean (μ)

DS	Deterministic	Auto-Configured		Non-Deterministic	
	i (sec)	t (sec)	Ratio	t (sec)	Ratio
1	224.11	0.03	7470	1.22	183.7
2	217.97	0.02	10899	6.27	34.8
3	239.22	0.03	7974	6.45	37.1
4	216.26	0.02	10813	6.43	33.7
6	282.42	0.07	4035	6.38	44.3
8	235.94	0.03	7865	6.25	37.7
9	249.68	0.04	6242	0.78	319.7
10	230.44	0.03	7681	6.34	36.3
11	246.37	0.03	8212	0.51	484.0
13	246.37	0.04	6159	0.94	263.2
14	622.88	0.29	2148	10.20	61.1
15	227.91	0.03	7597	0.46	497.6
16	360.73	0.11	3279	0.59	613.5
17	260.52	0.05	5210	0.77	338.8
μ	275.77	0.06	6827	3.83	213.25

showed good results. It recorded $\approx$ 94.5% average accuracy change and very good time of execution ratio of 6827; probably had fewer iterations.

Table VI: Jackknife Estimates for deterministic and non-deterministic search algorithms across 14 datasets (DS), and their mean (μ)

DS	Bias-Corrected Jackknifed Estimate		Confidence Interval			
			Deterministic		Non-Deterministic	
	Determ-inistic	Non-Determ-inistic	Lower	Upper	Lower	Upper
1	0.86	0.85	0.86	0.87	0.85	0.85
2	0.98	0.98	0.98	0.98	0.98	0.98
3	0.85	0.85	0.85	0.85	0.85	0.85
4	0.82	0.79	0.82	0.82	0.79	0.8
5	0.99	0.99	0.99	0.99	0.99	0.99
6	0.89	0.88	0.89	0.89	0.88	0.89
7	0.61	0.59	0.6	0.61	0.59	0.59
8	0.99	0.99	0.99	0.99	0.98	0.99
9	0.53	0.52	0.53	0.53	0.51	0.52
10	0.71	0.69	0.71	0.71	0.69	0.69
11	0.93	0.91	0.93	0.93	0.91	0.92
12	0.8	0.77	0.79	0.8	0.77	0.78
13	0.68	0.67	0.68	0.68	0.66	0.67
14	0.71	0.69	0.71	0.71	0.68	0.69
μ	0.81	0.80	0.81	0.81	0.80	0.80

F. Evaluation using Jackknife Estimation

Jackknife is used to evaluate the quality of the prediction of computational models. It uses resampling to calculate standard deviation error and estimate bias of a sample statistic, as shown in equations 3 and 4 [16]. We computed Jackknife across the 14 datasets and tabulated results as shown in Table VI. We recorded a zero for bias and standard errors across all datasets.

$$Var(\theta) = \frac{n-1}{n}\sum_{i=1}^{n}(\bar{\theta}_i - \bar{\theta}_{jack})^2, \quad \bar{\theta}_{jack} = \frac{1}{n}\sum_{i=1}^{n}(\bar{\theta}_i) \tag{3}$$

$$\bar{\theta}_{BiasCorrected} = N\bar{\theta} - (N-1)\bar{\theta}_{jack} \tag{4}$$

In Table VI we see different datasets record different values of Bias-Corrected Jackknifed Estimates. We also observe stable results are per the predictions in Table IV. Standard error is used for null hypothesis testing and for computing confidence intervals (upper and lower bounds). This explains why we observe confidence intervals deviating insignificantly. We also see the bias-corrected Jackknifed estimate deviating minimally because the standard error were zero across all the records. These results show that the non-deterministic search algorithm predictions are stable and reliable.

IV. Conclusion

In this research, we formulated a non-deterministic strategy in searching for the best hyperparameter in random forest algorithm considering number of trees, accuracy and time of searching hyper-parameter. The non-deterministic search strategy recorded significantly good results in maximizing accuracy, minimizing number of trees and minimizing searching time. Evaluations using Jackknifed Estimation show that its predictions are stable. Moreover, the non-deterministic search strategy had a significant accuracy levels and better utilization cpu processing and time in memory. This research can be widely adopted in algorithms hyperparameter search and in green computing to preserve computing resources.

ACKNOWLEDGMENT

We would like to express our appreciation to the French Embassy in Kenya and the French Government's Ministry of Foreign affairs for the financial support in this research.

REFERENCES

[1] J. Bergstra and Y. Bengio. "Random search for hyper-parameter optimization." Journal of Machine Learning Research, pp. 281-305, 2012.

[2] L. Breiman. "Random forests." Machine learning, Kluwer Academic Publisher, 45(1), DOI:10.1023/A:1010933404324, pp. 5-32, 2001.

[3] Breiman, L., and Cutler, A. (2003), "Random forests manual v4.0", Technical report, UC Berkeley. https://www.stat.berkeley.edu/~breiman/Using_random_forests_v4.0.pdf Date Accessed: July 2018.

[4] T. Chen and C. Guestrin. "Xgboost: A scalable tree boosting system." In Proceedings of the 22nd ACM sigkdd international conference on knowledge discovery and data mining, DOI:10.1145/2939672.2939785, pp. 785-794. ACM, 2016.

[5] I. Dewancker, M. McCourt, S. Clark, P. Hayes, A. Johnson and G. Ke. "A stratified analysis of bayesian optimization methods." Cornell University Library, arXiv:1603.09441 [cs.LG], 2016.

[6] Y. Ganjisaffar, T. Debeauvais, S. Javanmardi, R. Caruana and C.V. Lopes. "Distributed tuning of machine learning algorithms using MapReduce clusters." In Proceedings of the Third Workshop on Large Scale Data Mining: Theory and Applications, DOI:10.1145/2002945.2002947 USA, 2011.

[7] E. Hazan, A. Klivans and Y. Yuan. "Hyperparameter optimization: A spectral approach.", arXiv:1706.00764 [cs.LG], 2017.

[8] B.F. Huang and P.C. Boutros. "The parameter sensitivity of random forests." BMC bioinformatics, 17(1), DOI: https://doi.org/10.1186/s12859-016-1228-x, 2016.

[9] J.P. Lalor, H. Wu and H. Yu. "CIFT: Crowd-informed fine-tuning to improve machine learning ability". arXiv:1702.08563 [cs.CL], 2017

[10] Kenya Agricultural and Livestock Research Organization. www.kalro.org/. Date Accessed: July 2018.

[11] K. Senagi, N. Jouandeau and P. Kamoni. "Machine learning algorithms for soil analysis and crop production optimization: A review". In Proceedings of the International Conference on Mass Data Analysis of Images and Signals (MDA), USA, pp. 1-15, 2017.

[12] K. Senagi, N. Jouandeau and P. Kamoni. "Using parallel random forest classifier in predicting land suitability for crop production". Journal of Agricultural Informatics. Vol. 8 (3), 2017.

[13] J. Snoek, H. Larochelle and R.P. Adams. "Practical bayesian optimization of machine learning algorithms." In Advances in neural information processing systems, pp. 2951-2959, 2012.

[14] S.K. Smit and A.E. Eiben. "Comparing parameter tuning methods for evolutionary algorithms." In Evolutionary Computation CEC'09, IEEE, DOI:10.1109/CEC.2009.4982974, pp. 399-406, May 2009.

[15] T.P. Oshiro, S.J. Perez and A. Baranauskas. "How many trees in a random forest?" In Proceedings of the International Workshop on Machine Learning and Data Mining in Pattern Recognition, Springer, Berlin, Heidelberg, DOI: https://doi.org/10.1007/978-3-642-31537-4_13, pp. 154-168, 2012.

[16] S. Wager, T. Hastie and B. Efron. "Confidence intervals for random forests: The Jackknife and the infinitesimal Jackknife." Journal of Machine Learning Research 15(1), 1625-1651, 2014.