2020 2nd International Conference on Advances in Computing, Communication Control and Networking (ICACCCN) | 978-1-7281-8337-4/20/$31.00 ©2020 IEEE | DOI: 10.1109/ICACCCN51052.2020.9362840

Competitive Analysis of the Top Gradient Boosting Machine Learning Algorithms

Shyam R
Department of Computer Science & Engineering, The National Institute of Engineering, Mysuru, India 570008.
4NI17CS093_B@nie.ac.in

Sai Sanjay Ayachit
Department of Computer Science & Engineering, The National Institute of Engineering, Mysuru, India 570008.
4NI17CS014_B@nie.ac.in

Vinayak Patil
Department of Computer Science & Engineering, The National Institute of Engineering, Mysuru, India 570008.
4NI17CS107_B@nie.ac.in

Anubhav Singh
Department of Computer Science & Engineering, The National Institute of Engineering, Mysuru, India 570008.
4NI17CS012_B@nie.ac.in

***Abstract*—In this paper, we compare four state-of-the-art gradient boosting algorithms viz. XGBoost, CatBoost, LightGBM and SnapBoost. All these algorithms are a form of Gradient Boosting Decision Trees(GBDTs). They find wide usage across competitive machine learning contests like Kaggle, due to their flexibility and considerably faster training times. Since a typical vanilla GBDT is usually implemented as a black box model, our research makes an attempt to help improve the explainability of GBDTs. We performed an exhaustive 360 degree comparative analysis of each of the four algorithms by training and testing them on diverse datasets leveraging IBM's PowerAI^R AC922 CPU. The analysis was performed using two approaches; One was by training the baseline algorithms on the datasets, and the other was by performing systematic hyperparameter optimization (HPO) using the HyperOpt framework. Although the HPO process is resource-intensive, the Power System architecture facilitated lower training times without compromising the algorithm's accuracy for each of the datasets. We present the accuracy scores and training times across the four datasets for both the aforementioned approaches. The results imply that despite interesting trends observed across all the datasets, there is no clear winner that excels equally in every aspect of performance**

Keywords—Diverse Datasets; Exploratory Data Analysis; Gradient Boosting Algorithms; Hyper-Parameter Optimization; IBM PowerAI; Predictive Modelling

I. INTRODUCTION

Given the rapid increase in computing power and data driven approaches to tackle many of the real world problems today, Machine Learning (ML) has become an integral part of many SaaS(Software as a Service) solutions. Compared to several other ML techniques, gradient boosting seems to outshine others in many applications. Although techniques such as ensemble learning exist, gradient boosting algorithms have a sweet spot when it comes to faster training times and much higher accuracy. Tree based boosting algorithms are by far the most effective among them [1], and these algorithms are what we intend to examine in this paper.

The work presented in this paper was conducted as a part of the IBM Global Remote Mentorship Program.We, the team of Interns were assigned with the task of identifying and learning the key concepts of four top gradient boosting algorithms i.e. XGBoost, CatBoost, LightGBM and SnapBoost. Despite their different origins, all four iteratively perform gradient boosting; thus being labelled as GBDTs. GBDTs give substantial flexibility to the user to select and tune a wide variety of hyperparameters to suit their needs.

Two approaches exist for hyper-parameter tuning, one being tuning by hand through trial and error which is tedious and time consuming, and the other approach being techniques such as Bayesian Hyper-Parameter optimization (HPO) which automates the entire process of tuning [1]. For this purpose, we used an open source library called 'HyperOpt'[2]. In the experimental section we have provided the comparative results for both the cases. In order to analyze each of the algorithms from an overall perspective, each Intern was assigned with one algorithm and we worked on identifying the pros, cons and the key features that distinguish their chosen algorithm from the rest. Next we trained the algorithms on 4 diverse datasets including Numeric, Image, Temporal and Categorical data [3,4,5,6].

II. METHODOLOGY

In this section we describe the experimentation methodology adopted to carry out the analysis. We enlist the four datasets used for this task and also describe the key characteristics and ideas behind the four algorithms. We also present the results obtained on the IBM PowerAI^R AC922 Machine in the next section.

A. Datasets used for experimentation

- Numeric Dataset - The numerical dataset chosen was Pokemon_Numeric which contained 800 numeric entries and 11 Features with the objective to predict whether or not a pokemon is legendary, given the values of features such as HP, Attack, Defense Generation etc. 30% of the data was used for testing[3].
- Categorical Dataset - The categorical dataset chosen was Pokemon_Categorical which had a similar objective of the Pokemon_Numeric dataset but had 6 categorical features and 800 categorical entries. Similar to the numeric dataset, 30% was used for testing[4].

ISBN: 978-1-7281-8337-4/20/$31.00 ©2020 IEEE

- Image Dataset - The image dataset chosen was the CIFAR-10 dataset which has about sixty thousand 32x32 colour images distributed over 10 classes, with 6000 images per class.The train test split being 5:1 [5].
- Temporal Dataset - The temporal dataset chosen was Daily_Minimum_Temperatures_in_Melbourne dataset. It had 3650 entries comprising 3 years worth of data[6].

B. Algorithms used for experimentation

1) XGBoost: XGBoost is an ML algorithm based on decision-trees that uses a gradient boosting framework. It was a research project which was undertaken at the University of Washington [7]. Since then it has been credited with winning numerous Kaggle competitions.
The two main standouts for XGBoost are System Optimizations and Algorithmic Enhancements [7].

1. System Optimization:
   a. *Parallelization*: A parallelized implementation is used by XGBoost to approach the process of sequential tree building.
   b. *Tree Pruning*: Tree pruning is done backwards using the parameter 'max_depth'.

2. Hardware Optimization:
   This algorithm makes efficient usage of the hardware resources too.'Cache awareness' is a key concept involving the allocation of buffers internally in each thread so that it is possible to store statistics of the gradients which helps the algorithm accomplish this.

3. Algorithmic Enhancements:
   a. Regularization: It basically penalizes highly complex models through two methods: LASSO regularization (L1) and Ridge (L2) regularization which prevents overfitting [7,8].
   b. Sparsity Awareness: Through which the best missing value can be learnt automatically using 'sparse features'.
   c. Weighted Quantile Sketch: The WQS algorithm is employed by XGBoost within weighted datasets to find the optimal points of splitting.
   d. Cross-validation: The algorithm has a built-in cross validation method per iteration, which does away with the need to explicitly program this [7].

The steps involved in the boosting process are:

- We initialize the boosting algorithm with $G_0(x)$ [7,8]:

$$G_0(x) = argmin_\gamma \sum_{i=1}^{n} M(y_i, \gamma). \quad (1)$$

- Iteratively compute Gradient loss function:

$$r_{im} = -\alpha\left[\frac{\partial(M(y_i, G(x_i)))}{\partial G(x_i)}\right]_{G(x)=G_{m-1}(x)} \quad (2)$$

$$where\ \alpha\ is\ the\ learning\ rate$$

- The gradient of each step is fit by the $h_m(x)$.
- The multiplicative factor m is derived for each terminal node and $G_m(x)$ the boosted model is defined as [7,8]:

$$G_m(x) = G_{m-1}(x) + \gamma_m h_m(x) \quad (3)$$

2) LightGBM: LightGBM stands for Light Gradient Boosting Machine. It is a robust gradient boosting framework that is used for classification,ranking and many other ML tasks. It is a distributed, high-performance gradient boosting framework based on decision trees. The word 'Light' owes to its surprisingly fast nature. It was Developed by Microsoft Inc and the first stable release came out in January 2017.

The algorithm attempted to be a competitor to XGBoost in terms of both training time & accuracy[9]. The algorithm uses a relatively recent technique of Gradient-based One-Side Sampling (GOSS) in order to filter out the instances of data for finding a split value unlike XGBoost which uses histogram based & pre-sorted algorithms in order to compute the ideal split[9,10].

GOSS is implemented in such a way that it only keeps all the instances with large gradients and performs random sampling on the other instances. While the other boosting algorithms split the tree level-wise, LightGBM splits the tree leaf wise. It reduces loss more effectively than the level-wise algorithms and thus results in a better accuracy which can rarely be achieved by any of the other existing boosting algorithms. Thus, GOSS helps in achieving a great balance between minimizing the number of data instances whilst preserving the accuracy for learned decision trees. Light GBM grows the decision trees vertically while other algorithms grow trees horizontally. It will choose the leaf which has the maximum delta loss to grow. Leaf-wise algorithms can reduce more loss than a level-wise algorithm when growing on the same leaf. The formal algorithm for GOSS is given in [9].

3) CatBoost: CatBoost is a recent open-source ML algorithm from Yandex. It can easily be integrated with modern deep learning frameworks such as Google's TensorFlow or Apple's Core ML. It performs well with diverse data types to help solve a wide range of problems that businesses face today, making an ideal candidate for this analysis. CatBoost is also appreciated for its ability to outperform all the algorithms when it comes to handling categorical features [11].

Normally One-Hot Encoding is done for categorical features but this inturn incurs the dimensionality curse. Earlier algorithms tackled this sparse features issue using Exclusive Feature Bundling. The way categorical features are dealt by

Catboos is through the generation of random permutations of the data and for each such permutation, it computes the mean of label value for the sample with the matching category value which is present ahead of the given one in the permutation[11,12]. The algorithm also is supposedly 13–16 times quicker at execution or predictions than the other libraries as per the Yandex benchmark.

Before each split is selected in the tree, categorical features are transformed to numerical. The transformation of categorical to numerical features is done by the following 3 step method[11,12]:

1. Random order permutation of the set of input objects.

2. Label value is converted from a floating point into an integer.

The given procedure depends upon the ML problem being solved which is in turn based on the loss function selected.

All categorical feature values are transformed to numerical using the following formula:

$$avg\_target = \frac{countInclass + prior}{totalCount + 1} \quad (4)$$

where $CountInClass$is how many times the label value was equal to "1" for objects with the current categorical feature value[12]. $totalCount$- is the total number of objects (up to the current one) that have a feature value matching the current one[11,12]. $prior$- is a number (constant) defined by the starting parameters.

4) SnapBoost: SnapML is a ML library developed by IBM Research, Zurich and it offers a boosting machine, SnapBoost, that can be used for classification and regression. The boosting functionality is available using the class BoostingMachine() [13,14].

On comparing the performance of other classification libraries on TensorFlow on Google Cloud with SnapML, the Logistic regression classifier of the SnapML library trained in 91.5 seconds, which is 46x faster than the best result reported so far. The dataset on which this was implemented is Criteo Terabyte dataset released by Criteo which is Industry's Largest-Ever Dataset for Machine Learning to Academic Community[13].

SnapBoost performs a functional gradient descent to learn an ensemble of decision trees. The boosting machine does not make a heterogeneous ensemble of decision trees, instead takes a probabilistic approach to select the maximal tree depth at each boosting iteration. This depth can be controlled by the user according to a uniform distribution; tuning which can lead to achieve better generalization accuracy compared to other boosting frameworks on some datasets [14]. The fit and predict functions accept numpy.ndarray data structures.

Similar to XGBoost, the BoostingMachine also supports both example-wise and feature-wise subsampling at each boosting round and also L2-regularization[14].

We initially utilized several Exploratory Data Analysis (EDA) techniques to identify feature-importance, correlation of features and also performed feature engineering on the datasets. All of this along with numerous data visualizations have all been neatly provided in a Jupyter notebook for each dataset in our public repository. We have trained each of these four algorithms across all the four datasets by choosing appropriate train-test splits; necessary data cleaning and preprocessing was also done. The consolidated results are given next.

III. EXPERIMENTAL RESULTS

We trained XGBoost, LightGBM and SnapBoost with the 4 datasets on the CPU on the IBM Power System AC922 server with and without the usage of Hyperopt (for HPO).The same was done with CatBoost on Google Colab CPU as Anaconda ppc64le on the PowerAI AC922 server doesn't support CatBoost. We plot the Accuracy (in %) achieved by the algorithm and the training time (in seconds) obtained in all the cases.

HyperOpt adopts Tree of Parzen Estimator(TPE) for Hyperparameter tuning. For the purpose of illustration of how HyperOpt works, we have provided the HyperOpt iteration graphs (Accuracy Vs. HyperParameter Value) for several Hyper-parameters (HPs).

These graphs will enable the reader to better understand the range of values through which Hyperopt iterated for each HP. It is also helpful in determining the optimal value of a HP, for which the accuracy of the model was the highest. The chosen optimal HP value is highlighted with Red color in the graphs.

Trials were conducted for tuning several HPs for the 4 algorithms in all the datasets. Owing to the constraint on manuscript length we have provided only the graphs of Numeric Dataset here and not any other dataset. The reader can feel free to check the graphs for all the datasets and algorithms on our Github Repository.

We provide the code and all the graphs for our experiments here: https://github.com/anubhav48/Catboost-vs-Lightboost-vs-XGBoost-vs-Snapboost

More information on the run-time environments we used can be found in [15,16].

A. *Performance on the Numeric Dataset*

1) Accuracy Plots:

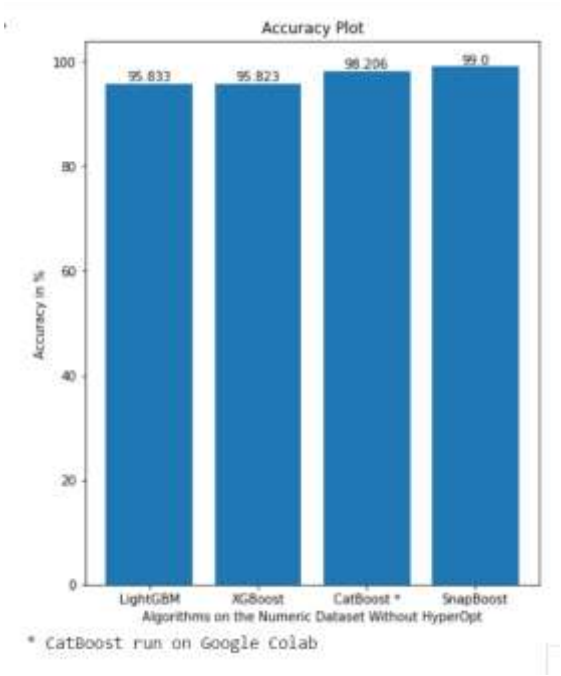

Fig 1. Accuracy Plot for Numeric Data without HPO

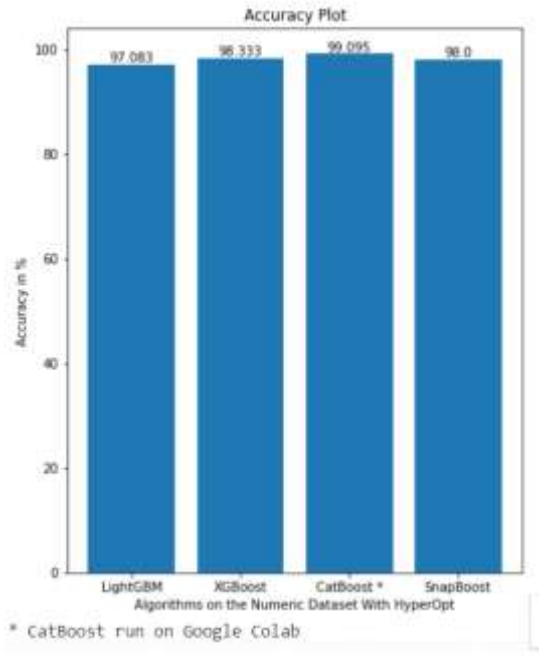

Fig. 2. Accuracy Plot for Numeric Data with HPO

Without HPO: On the Numeric dataset, SnapBoost performed very well with 99% accuracy in predictions, followed by CatBoost with 98% and LightGBM with 95.83% and XGBoost followed closely with 95.82% which all are good performances without any optimizations.

With HPO using HyperOpt: While CatBoost and SnapBoost had more or less the same accuracies here, LightGBM and XGBoost showed an improvement of 2-3%.

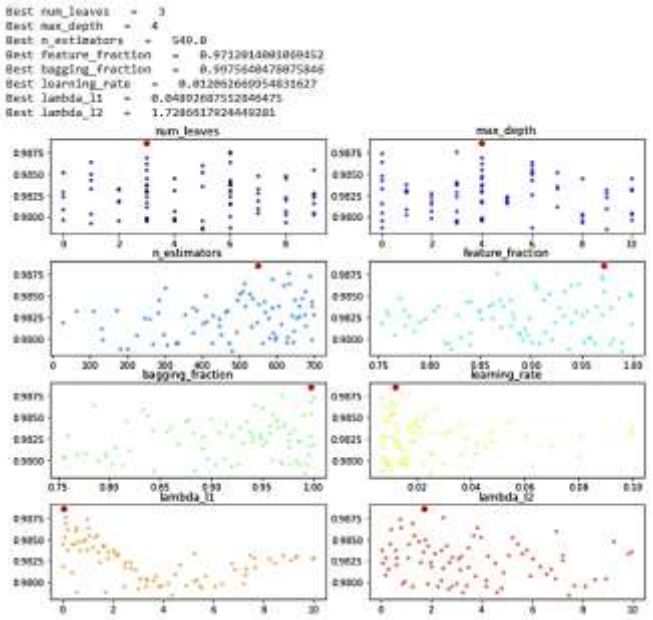

Fig. 3. Accuracy Vs HP Graph for LightGBM

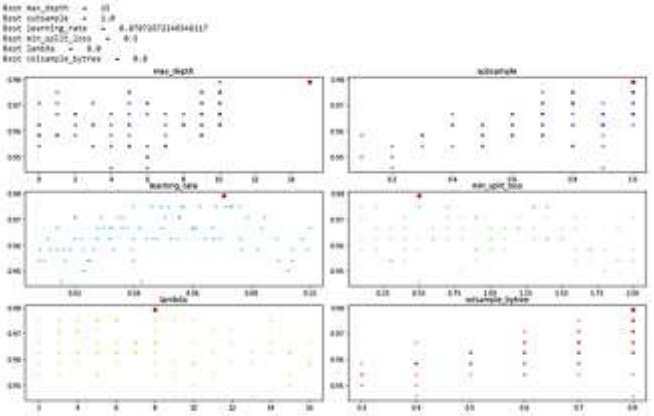

Fig. 4. Accuracy Vs HP Graph for XGBoost

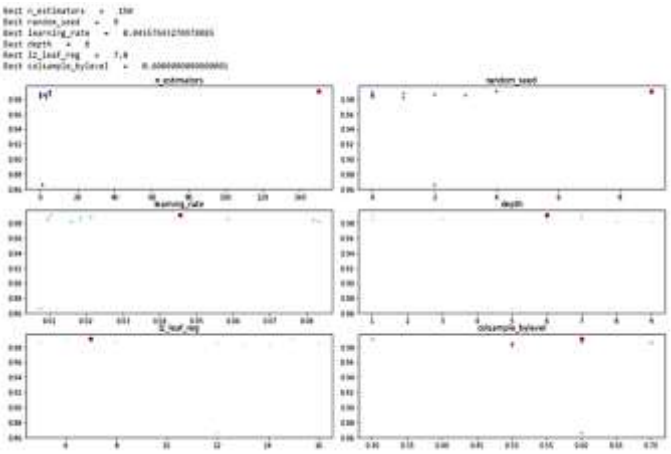

Fig. 5. Accuracy Vs HP Graph for CatBoost

The tuning process was carried out for several HPs for all four algorithms individually. The best suited or optimal combinations of HPs are auto-selected by Hyperopt and we have highlighted them in red color.

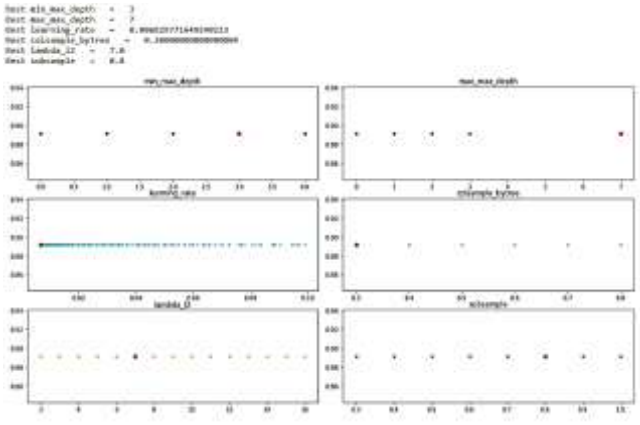

Fig. 6. Accuracy Vs HP Graph for SnapBoost

2) *Training time Plots:*

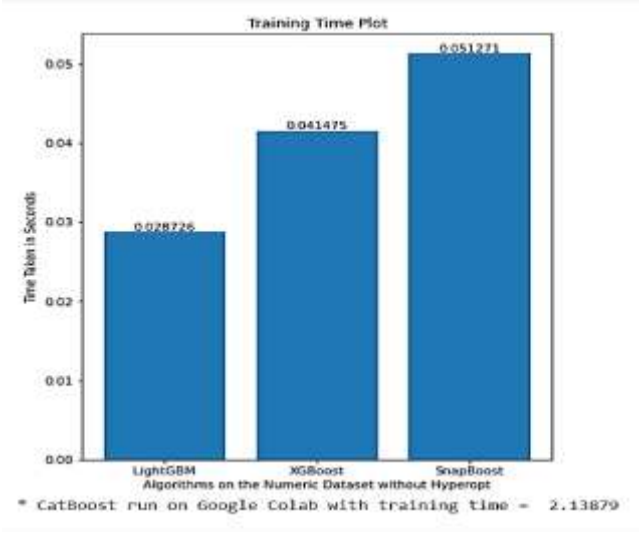

Fig. 7. Training Time Plot for Numeric Data without HPO

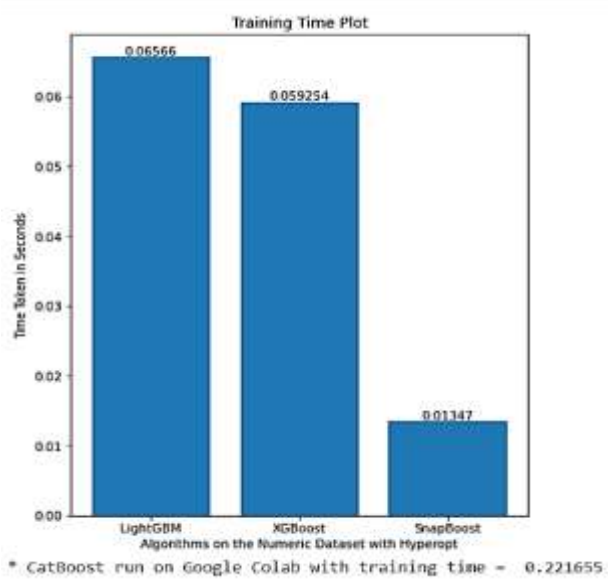

Fig. 8. Training Time Plot for Numeric Data with HPO

B. Performance on the Categorical Dataset

Without HPO: On the categorical dataset, Catboost outperformed the rest by achieving a 99.5% accuracy, the second place is taken by SnapBoost(98%) while XGBoost(97.24%) that narrowly overtakes LightGBM (96.68%) gives a tough competition.

With HPO using HyperOpt: Using Hyperopt, All the algorithms showed an improvement in performance, SnapBoost improved by 1% while the others improved by 2%.

C. Performance on the Temporal Dataset

Without HPO: On the temporal dataset, LightGBM beats the rest and achieves the highest accuracy of 54.45% while CatBoost achieves a 54.12% score. XGBoost fails miserably.

With HPO using HyperOpt: SnapBoost showed the most promising improvement by 27% increase in accuracy. While all others showed an improvement of around 13-16% in the accuracy.

D. Performance on the Image Dataset

Without HPO: On the image dataset CatBoost is the clear winner as it achieves an accuracy of 61% while SnapBoost not far behind manages to achieve a 50% accuracy. LightGBM again, fails here with a mere accuracy of 10%.

With HPO using HyperOpt: XGBoost showed an improvement of 13% increase in the accuracy while others showed only a marginal change.

E. Inferences

On examining the performance of algorithms on the 4 diverse datasets we infer that the most consistent performance in terms of accuracy and training time was seen in SnapBoost & XGBoost across all the datasets. XGBoost always stood second or third in terms of training time, and put up a stiff competition with the other algorithms in terms of accuracy. CatBoost worked very well for Categorical data and outperformed the other algorithms in Image Data as well,but was the slowest algorithm in 3 out of 4 cases. LightGBM is a very fast algorithm in terms of training on all datasets excluding image data. But it's training time prowess is marred by its inconsistency in accuracy. SnapBoost showed good performance on the temporal and numeric dataset and it is consistent in terms of accuracy. The huge highlight being that SnapBoost when run on the IBM Power System AC922 server with HPO is the fastest algorithm in terms of training time; which is then followed by LightGBM.

We have observed significant improvement in the accuracies of all the algorithms after using HyperOpt for Hyper Parameter Optimization. It reduced overfitting in some of the cases while in others it helped in making better fits to the data. For convenience we provide a tabular representation of the final consolidated accuracies below.

TABLE I. ACCURACY OBTAINED WITH HYPER-PARAMETER OPTIMIZATION FOR EACH DATASET (IN %)

Dataset	Algorithms			
	LightGBM	***XGBoost***	***CatBoost****	***SnapBoost***
Numeric	97.083	98.333	99.095	98.0
Categorical	98.755	99.17	97.421	99.0
Temporal	66.855	53.0	67.355	75.64
Image	14.0	41.66	61.836	50.0

*CatBoost Run on Google Colab

IV. CONCLUSION

We have presented an in-depth experimental analysis of four state-of-the-art GBDT packages: XGBoost, LightGBM Catboost and SnapBoost. These algorithms were validated for being robust against overfitting, ability to handle categorical variables, and multimodal data while leaving a small memory footprint. Our research aids in improving the explainability of GBDTs across several use cases. Although we can infer from the results that all the four algorithms are equally competent, yet SnapBoost coupled with the PowerAI system is able to obtain better accuracy than XGBoost in all the 4 datasets even before using HPO. After using HPO technique, SnapBoost performed better in two cases and in the other two it was pretty close to the result obtained by XGBoost.In terms of training time, SnapBoost in general performed the best on IBM Power Systems compared to the other two frameworks which were run on the same platform. Despite extensive research in this domain, the challenge of building a robust GBDT framework that excels in all scenarios still continues to be an open problem. We hope that our research has substantially aided in the future enhancements of the on-going work around GBDTs.

ACKNOWLEDGMENT

We would like to extend our sincere gratitude to our Project guide Mr. Sangeeth Keeriyadath [17] (Advisory Software Engineer (Watson distributed machine learning frameworks) at Systems Development Lab, IBM) under whose guidance these experiments were conducted. We would also like to thank our faculty mentor Dr. Abhinandan S.P [18] (Associate Professor, Department of Computer Science & Engineering, NIE Mysuru) for his valuable insights and efforts. We are deeply indebted to IBM Watson for giving us this opportunity as part of their Global Remote Mentorship Program [19].

REFERENCES

[1]. A. Anghel, N. Papandreou, T. Parnell, A. De Palma, and H. Pozidis, “Benchmarking and optimization of gradient boosting decision tree algorithms,”arXiv preprint arXiv:1809.04559, 2018.
[2]. “http://hyperopt.github.io/hyperopt/.”
[3]. “https://www.kaggle.com/mlomuscio/pokemon.”
[4]. "https://www.kaggle.com/rounakbanik/pokemon.”
[5]. “https://www.cs.toronto.edu/ kriz/cifar.html.”
[6]. “https://www.kaggle.com/paulbrabban/daily-minimum-temperaturesin-melbourne.”
[7]. 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, pp. 785–794, 2016.
[8]. "https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guideto-understand-the-math-behind-xgboost/.”
[9]. G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, “Lightgbm: A highly efficient gradient boosting decision tree,” in Advances in neural information processing systems, pp. 3146–3154, 2017.
[10]. “https://lightgbm.readthedocs.io/en/latest/.”
[11]. https://towardsdatascience.com/catboost-vs-light-gbm-vs-xgboost-5f93620723db.”
[12]. A. V. Dorogush, V. Ershov, and A. Gulin, “Catboost: gradient boosting with categorical features support,”arXivpreprint arXiv:1810.11363, 2018.
[13]. “https://www.zurich.ibm.com/snapml/.”
[14]. “https://ibmsoe.github.io/snap-ml doc/v1.6.0/manual.html#snapboost.”
[15]. “https://www.ibm.com/support/knowledgecenter/power9/p9hdx/8335_gtg_landing.htm.”
[16]. “https://colab.research.google.com/notebooks/intro.ipynb.”
[17]. “https://www.linkedin.com/in/sangeek”
[18]. “https://nie.ac.in/faculty/abhinandan-s-p/”
[19]. “http://connecttobuild.in”