# A Framework for Searching a Predictive Model

Yoshiki Takahashi
Tokyo Institute of Technology
takahashi.y.bz@m.titech.ac.jp

Masato Asahara
NEC System Platforms Research Labs
masahara@nec-labs.com

Kazuyuki Shudo
Tokyo Institute of Technology
shudo@c.titech.ac.jp

## 1 INTRODUCTION

In data analytics with machine learning (ML) technology, data scientists have to explore massive number of possibilities of predictive model designs to build highly-accurate predictive models for each prediction problem because there is no almighty predictive model available as no free lunch theorem [12] shows. The possibilities of model designs can be classified into three major aspects: 1) algorithm choice, 2) hyper parameter tuning of each ML algorithm and 3) preprocessing strategies for training data such as missing value imputation policies, random sampling ratio and convert too fine category information into generalized one, e.g., generate state name information from address information, and so on. Although many approaches to efficiently completing the search of model design possibilities have been proposed [3, 4, 9, 11], data scientists essentially have to spend much time on the work on model design because it is essentially a non-convex optimization problem. To the best of our knowledge, no frameworks which assist not only algorithm choice and hyper parameter tuning but also data preprocessing options are proposed.

To release data scientists from the issue, we are developing a novel framework for automatic model design. This paper introduces our idea which enables data scientists to quickly find a predictive model which achieves enough high accuracy with no 'hand-made' operations including data preprocessing part. Our contributions in this work are techniques that enable high-speed exploration of model design possibilities. They are summarized below.
1) A distributed computing framework: It leverates both in-memory computing platforms such as Apache Spark [13] and highly-tuned machine learning implementations such as TensorFlow [2] and XGBoost [5]. Not only can it do algorithm choice and hyper-parameter tuning, but it can explore preprocessing strategies unlike existing frameworks. The preprocessed data should be fed to training in a no-memory-copy manner though it is in deveropment.
2) Low-frequent data collection for ML executions: High freqent data communication induces much wait time to ML algorithm executions. To avoid the increase of data communication, our framework employs low-frequent data collection mechanism which keeps the frequency to be constant to the number of preprocessing strategies.
3) Task scheduling based on predicted training times for ML configurations: The execution time of each worker varies heavily depending on a ML configurations, that is a set of algorithm and its hyper-parameters. Our framework assigns a set of cofigurations to a worker so as to minimize the execution time of the longest-lasting worker by estimating the completion time of ML algorithm execution with the prediction model for the time.

Our preliminary evaluations demonstrate that our framework automatically produces an enough highly-accurate predictive model in reasonable time. The results show that the predictive model generated in the framework achieves higher accuracy of prediction than that of simple predictive model designs in around 15 hours with 32 CPU cores even if the cumulative time of the model designs is several days.

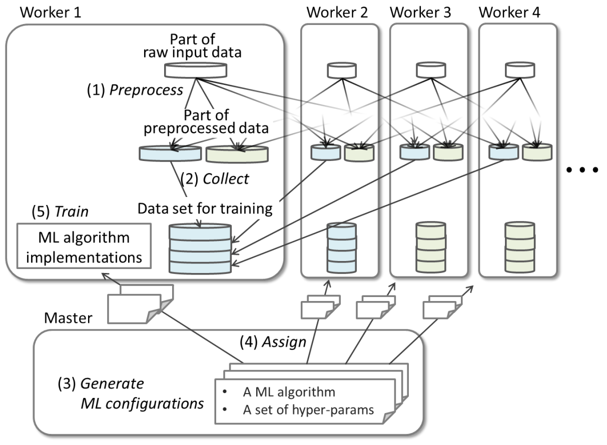

**Figure 1: Our framework works as follows: (1) collect preproccesed data from all workers, (2) generate machine learning configurations, (3) assign a set of configurations to workers, (4) train predictive models with the configurations.**

## 2 RELATED WORK

Several pieces of prior work [3, 4, 9, 11] present the generation algorithm of hyper parameter candidates. It is a part of future work to incorporate them with our framework.

Our work is closely related to HyperDrive [8] and TuPAQ [10], both of which focus on the exploration of model designs. In contrast, our framework can also explore pereprocessing strategies with low-frequent data collection described in Section 3.1. Additionally, we are trying to eliminate memory copy from preprocessing.

## 3 FRAMEWORK DESIGN

Figure 1 illustrates an overview of the proposed framework. The process of the framework consists of two major parts: data preprocessing part for training data and execution part for ML algorithm implementations. The data preprocessing part executes variety of data preprocessing strategies on raw input data such as different missing value imputation policies and different feature engineering strategies. And then, it generates training data from the preprocessed input data. The data preprocessing part utilizes the functionality of the distributed computing system such as Spark SQL and Spark ML. On the other hand, the execution part for ML algorithm implementations runs variety of ordinary implementations of ML algorithm such as Google TensorFlow and XGBoost in parallel with different hyper parameter configurations on the same distributed computing system as the data preprocessing part was performed. It utilizes the parallel computation functionality of the distributed computing system, e.g., Map task execution of MapReduce computing systems [6]. Between the two parts, the framework converts the format of the training data from the one for the distributed computing system such as RDD [13] to the one for ML algorithm implementations such as a Java matrix object in single worker's memory.

Because the number of combinations of ML algorithm implementations and their hyper paramters becomes several thousands order, the major computation part of the framework is the execution part of ML algorithm implementation. To accelerate the part, the framework employs two techniques: low-frequent data collection and prediction-based task scheduler.

## 3.1 Low-frequent data collection

Since the ordinary ML algorithm implementations utilize shared-memory architecture, training data in the distributed memory on the distributed computing system has to be collected to the memory of a worker which executes the ML algorithm implementations. If the framework performs the collection of preprocessed data on every workers at every execution of data preprocessing strategies, $O(lw)$ times of data commucation is kicked, where $l$ is the number of preprocessing strategies and $w$ is the number of workers.

To avoid the linearly-increased overhead to $l$, the framework employs low-frequent data collection mechanism. The mechanism first performs data preprocessing by using all workers and store the resulted data into the distributed memory among workers. Second, it assigns $\frac{w}{l}$ workers for each preprocessing strategy. Third, each worker collects preprocessed data based on the strategy the worker is assigned for, and converts the data with a compatible format for ML algorithm implementation. Forth, the framework distributes execution tasks of ML configurations, that is a set of algorithm and hyper-parameters, to each worker in the same-strategy-hold $\frac{w}{l}$ workers. Finally, the framework starts the execution of the task on every CPU cores of workers in parallel. This design reduces the frequency of data collection to $w$, which is constant to increase of preprocessing strategies.

## 3.2 Task scheduling based on predicted training time

To minimize the execution time of the longest-lasting worker, our framework employs task scheduling based on predicted times times for ML configurations. The scheduler first estimates the completion time of each combination of ML algorithm and hyper parameter configuration by using a pre-learned preditive model for the completion time. The scheduler then calculates a task assignment plan which approximately balances the completion time of tasks assigned to a worker by solving a bin-packing problem with the estimatied completion time. The key idea of this task scheduling is to assign a task from longer one in a round-robin fashion on the basis of the estimated completion time. Current prototype utilizes a linear regression model as the predictive model for the completion time of ML algorithm.

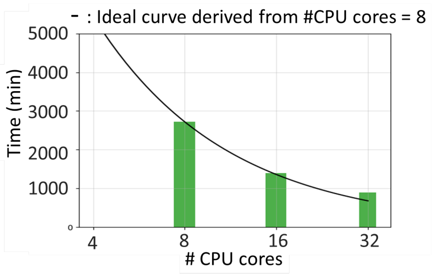

**Figure 2: Completion time of model design.**

**Table 1: Test accuracy of the best model obtained by each method.**

| | Classification error | AUC |
|---|---|---|
| **Proposed framework** | **0.129** | **0.862** |
| Gradient boosting tree of XGBoost | 0.131 | 0.860 |
| Logistic regression | 0.136 | 0.714 |

## 4 PRELIMINARY EVALUATION

We implemented the prototype of the framework using Apache Spark 1.6.0 and performed preliminary evaluation to demostrate if the framework can produce an enough high-accurate predictive model at high speed for the number of workers. The prototype used the ML algorithm implementation for the multi-layer perceptron model of TensorFlow and that for the gradient boosting tree model of XGBoost. In the evaluation the framework explored 495 and 480 combinations of grid-based hyper parameter search space for the multi-layer perceptron and the gradient boosting tree, respectively. In the evaluation we used the public competition data for KDDCUP 2015 [1]. The workers used in the evaluation employed up to two 2.5GHz CPU cores and 128GB memory.

Figure 2 shows the completion time of model design for number of CPU cores. The result demonstrates that it was completed to explore 975 possibilities of model designs among learning of multi-layer perceptron models and gradient boosting tree models in around 900 minutes, which lasted several days when we ran the exploration serially. The result also shows that our framework achieved almost linear performance improvement to the number of CPU cores. This suggests that the framework has a potential to increase the execution performance by simply adding workers.

Table 1 compares the best accuracy of the framework and the accuracy of a gradient boosting tree model learned with XGBoost and a logistic regression model learned with scikit-learn [7]. The hyper parameters used in the learning of the gradient boosting tree model and logistic regression model are default configurations of XGBoost and scikit-learn, respectively. The result shows that the framework achieved the highest accuracy in both of classification error and AUC.

## 5 CONCLUSION

This paper presents our framework for automatic model design, which enables data scientists to build a highly-accurate predictive model at high speed with no 'hand-made' operations.

## REFERENCES

[1] 2015. KDD CUP 2015. (2015). https://biendata.com/competition/kddcup2015/.

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2016).

[3] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13, Feb (2012), 281–305.

[4] James Bergstra, Dan Yamins, and David D Cox. 2013. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in Science Conference*. 13–20.

[5] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM, 785–794.

[6] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.

[7] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12, Oct (2011), 2825–2830.

[8] Jeff Rasley, Yuxiong He, Feng Yan, Olatunji Ruwase, and Rodrigo Fonseca. 2017. Hyperdrive: Exploring hyperparameters with POP scheduling. In *Proceedings of the 18th International Middleware Conference, Middleware*, Vol. 17.

[9] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*. 2951–2959.

[10] Evan R. Sparks, Ameet Talwalkar, Daniel Haas, Michael J. Franklin, Michael I. Jordan, and Tim Kraska. 2015. Auttomating Model Search for Large Scale Machine Learning. In *Proceedings of the Sixth ACM Symposium on Cloud Computing (SoCC '15)*. ACM, New York, NY, USA, 368–380. https://doi.org/10.1145/2806777.2806945

[11] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '13)*. ACM, New York, NY, USA, 847–855. https://doi.org/10.1145/2487575.2487629

[12] David H Wolpert, William G Macready, et al. 1995. *No free lunch theorems for search*. Technical Report. Technical Report SFI-TR-95-02-010, Santa Fe Institute.

[13] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster computing with working sets. *HotCloud* 10, 10-10 (2010), 95.