CS 519: Applied Machine Learning **(Due: June 8th, 2020)**

Submission Homework 5

Instructor: Prof. Liang Huang *Name:* Rahul Damineni, *ONID:* 933-922-122

XGBoost: A Scalable Tree Boosting System[1]

Non-technical background

1. 'XGBoost: A scalable tree boosting system'[1] is a paper published by **Chen Tianqi, and Carlos Guestrin**.

   Tianqi Chen received his Ph.D from Paul G. Allen School of Computer Science & Engineering, University of Washington. The co-author of this paper, Carlos Guestrin (PI), is Tianqi's advisor.

2. This paper was published in the Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2016/8/13)[1]. It is one of the most popular papers in this field with 5830 citation to date. The citation trend is increasing with every year with nearly half of the total citations are made in the year 2019[2]. This paper received several notable awards such as John Chambers award[3] and High Energy Physics meets Machine Learning award[5].

3. The paper used a combination of custom and publicly available data sets for bench-marking. AllState, Criteo and Higgs Boson datasets are randomly partitioned into training and development splits while Yahoo! LTRC[6] is a popular open-sourced data set for learning to rank algorithms. The official train test splits were used for experimentation. The implementation **XGBoost** is open-sourced[7] too. There's a demo included as a part of source-code. And there are several talks by the author discussing the paper. All these factors contributed to its strong reproducibility. It is a popular algorithm that powered most winning solutions in recent Machine Learning challenges[8].

Core *(what-why-how-wows)*

1. **(what)** The most signification contribution from this paper is a way to train a large scale tree boosting model that leverages on petabytes of data but with minimum resources. The scalability problem is first addressed in the paper. The scalability is achieved by improving speed, regularisation and resource utilization problems. The complementary ideas from statistics, data encoding, computer architecture are borrowed from existing literature but tightly integrated with each other to make an efficient system which is a good trade-off between scale, expressibility and compute power.

2. **(why)** Data is growing exponentially! A general rule of thumb (and in life) is the more data your model comprehends, the better it performs. So, machine learning algorithms that can seamlessly consume such a volume of data will always perform better than the ones that can't. And it's not always practical to train a model on a large data set. GPU is expensive – so neural networks, although algorithmically scalable, are not computationally so. Memory is costly too – which is why we need to leverage on comparatively cheaper pair of options: disk and CPU. That led to a this highly optimised system. The closest functional alternative, Gradient Boosting is not only worse at generalisation but takes magnitudes longer than XGBoost to fit on a data set of comparable size. It is not even possible to train on a large scale data set that doesn't fit in memory.

3. **(how)** To generalise the learning, many implicit and explicit regularisation techniques were used.

   - As a first step, a regularisation term penalizing leaf weights to curate simpler and predictive functions for the model.

   - Since Gradient Tree Boosting is a class of additive optimisation algorithms, shrinkage helps in reducing the influence of individual trees and provides scope for adding more function which would result in better generalisation of the model.

   - Tree specific regularisation techniques and other techniques such as row and column sub-sampling were are also discussed.

   Regularisation improves model's accuracy while not compromising expressivity.

   To speed-up learning, first, data properties are exploited.

   - One key optimisation was to selectively pick split candidates instead of brute-forcing on all possible values across all features (exact greedy algorithm). This was possible through weighted quantile sketch – which locally/ globally suggests the split candidates based on their **relative** impurity measures. This is approximate split finding algorithm.

   - Sparsity-aware split finding works for both exact greedy and approximate algorithm. The idea is to predict which branch a missing value would likely to take based on trend observed on non-missing data. This idea enables general automated imputation and eases computation complexity to linear in number of non-missing entries in the input.

   And then system related optimisations were performed to better accommodate the above optimisations.

   - Column blocks for parallel learning: To reduce the cost of sorting, data is stored as in-memory units called "blocks". With-in the block, the data is stored in the compressed column format. Each column will be sorted based on their values at the beginning so this order can be reused during the iterative process. For both exact and approximate algorithms, split candidate selection becomes linear. These blocks can be parallelized between threads.

   - Out-of-core computation: Dividing the data into blocks meant we don't have to load the entire data set into memory at once. We have independent threads loading these blocks into memory for future computation of gradients. Even with that, the disk read operation is too expensive and usually be the bottle-neck. To improve this bottle-neck and trade some disk reading cost with computation, the authors took two standard approaches:
     - Compression: compressing columns would reduce the I/O volume on disk while increasing the compute for compressing/decompressing the data.
     - Sharding: Block sharding is distributively storing a single block across multiple disks so different threads can parallely read and assemble the block for computation. This will help reduce the block read/write time as assembling cost is usually very low.

   - Avoiding Cache-misses: based on our split candidate selection algorithms, we would be loading same gradients repeatedly from memory into CPU across the row index. These are not stored in a contiguously so that would mean a high cache-miss rate. The natural solution for exact algorithm would be to have a buffer of all relevant gradients so cache misses aren't as expensive. For approximate algorithm, the right block size that balances the cache property and parallelisation was found to be 2^{16}.

4. **(wow)**

   - Weighted quantile sketch provided a theoretical guarantee while existing approximate split candidate suggestion algorithms are heuristics based and would resort to sorting on a random subset of data. Approximate algorithm with a small enough global epsilon behaved like an exact algorithm.

   - On a sparse data set, the proposed sparisity aware split finding algorithm is at least 50 times faster than the version that doesn't take sparsity into consideration.

   - Using cache aware pre-fetching on a large data set improved the performance by a factor of two.

   - Unlike pGBRT, XGBoost scaled linearly with number of cores due to its column block design

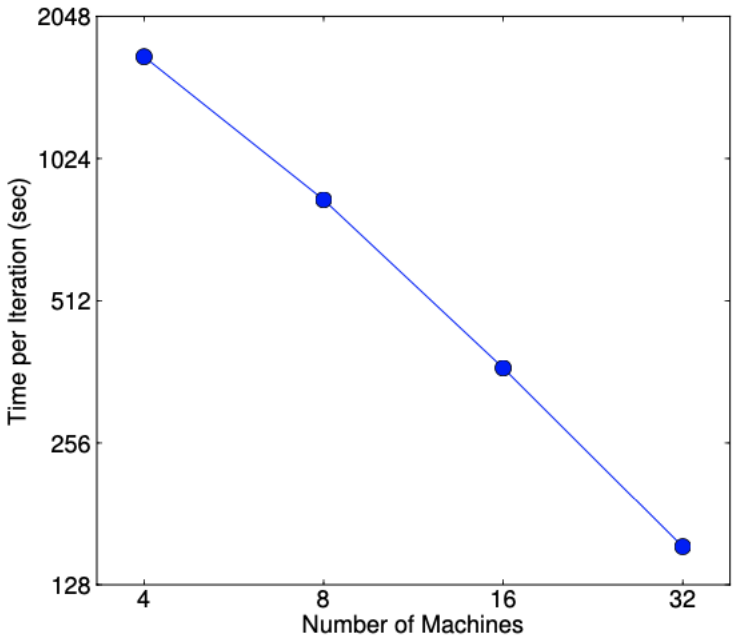

Figure 13: Scaling of XGBoost with different number of machines on criteo full 1.7 billion dataset. Using more machines results in more file cache and makes the system run faster, causing the trend to be slightly super linear. XGBoost can process the entire dataset using as little as four machines, and scales smoothly by utilizing more available resources.

Scope, limitations and enhancements

- I wasn't convinced that sparsity aware split finding results in truly general imputation method – mainly because of the absence of empirical studies.
- I would have learn something about the feature distribution and somehow made use of it to improve split candidate recommendation algorithm considering the current iteration.

- Overall, the paper felt like a review of highly engineered system. I liked the way the authors considered every component of process – the whole pipeline of training – starting from data preparation on the disk to algorithmic execution on CPU.
- It is surprising to see a two factor improvement in performance by optimising caching which isn't even exposed anywhere in the direct application pipeline.
- My key takeaway from this paper is to use gradient boosting methods when your data is irregular. Use XGBoost to scale by spending the minimum resources. Figure out the optimal block size while using approximate algorithm depending on what hardware I am running.
- My philosophical take-away from this paper is how to think about optimising a system: the process is to layout design options available based on what attributes of the system we want to optimise. While coming up with design options, be granular and try anything that seem reasonable – even when it feels beyond scope as the goal is to optimise the system attributes without limiting ourselves.

Relevance

1. Generalisation: Concepts like shrinking and weight were introduced in the class and I appreciated the scope of generalisation in a real world system.
2. Iterative algorithm and accuracy metrics: they are relatable to what I've learned in class during assignments and lectures.
3. I had to learn Statistic concepts like:
   - What is tree boosting method? What is impurity index? How CARTs are built?
   - How quantiles represent spread of population?
   - What is additive function optimisation and why use second order approximation, why not stick with the first order?
   - What is weighted quantile sketch and how is it helping select better candidates during local approximate split candidate algorithm?
4. Other than that, I had to learn system engineering concepts like:
   - What is cache-miss and how it influences execution time of an algorithm?
   - How CPU I/O is connected to disks and where's the true bottle-neck: CPU-bound task Vs disk-bound task

References

[1] Chen, Tianqi, and Carlos Guestrin. *XGBoost: A Scalable Tree Boosting System* Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2016): n. pag. Crossref. Web.

[2] Google Scholar citation history `shorturl.at/isw25`

[3] John Chambers award winner: Tong He (Simon Fraser University) and Tianqi Chen (University of Washington) `http://stat-computing.org/awards/jmc/winners.html`

[4] When High Energy Physics meets Machine Learning Award – 2016 `https://higgsml.lal.in2p3.fr/prizes-and-award/award/`

[5] XGBoost source code `https://higgsml.lal.in2p3.fr/prizes-and-award/award/`

[6] O. Chapelle and Y. Chang. *Yahoo! Learning to Rank Challenge Overview.* Journal of Machine Learning Research - W CP, 14:1–24, 2011.

[7] XGBoost open-sourced GitHub repository `https://github.com/dmlc/xgboost`

[8] Most popular ML algorithms for winning Kaggle competitions `https://www.kaggle.com/general/25913`