A Survey Into Modern Gradient Boosting Machine Implementations

A**** P*********

Spring 2020

Abstract

When considering predictive regression and classification problems on tabular data, Gradient Boosting Machines have effectively gained the position of an universal approximator among Data Science practitioners. They are ensemble models that require little preprocessing, scale well and are relatively robust to outliers and missing data. First, this paper introduces Gradient Boosting Machines through a brief survey into the history of boosting and gradient boosting, including the improvements later added to them. Then, it presents three different implementations of the Gradient Boosting Machine: LightGBM, XGBoost and NGBoost. The first two propose novel techniques to deal with issues such as high dimensionality, sparsity and high data count. The third one aims to produce probabilistic predictions and consequently presents a method for simultaneous multi-parameter boosting. This paper focuses on mathematical aspects of these techniques and seeks to evaluate their merits.

1 Introduction

The *boosting* algorithm AdaBoost, as presented by [FS97], can be considered a prototype of modern Gradient Boosting Machines. The algorithm aims to maintain a set of weights $\mathbf{w}$ over the observations $\mathbf{x}$ in the training data. The *weak learners*, such as decision tree functions, are fit to the data adjusted by the current weights. They ultimately form a basic additive model $F(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^{M} \beta_m h_m(\mathbf{x})\right)$. On each iteration, the error of the newly added decision tree is calculated and the weights are adjusted to minimize it. The weights are increased for observations misclassified by the current weak learner and decreased for the correctly classified observations. The predictions of the individual weak learners are combined, or *boosted*, into a strong learning algorithm. The resulting algorithm produces predictions by a weighted majority vote of the weak learners.

Actual *gradient boosting*, as proposed by [Fri01], also works by fitting simple, additive functions. However, now the weak learners are iteratively fit to the *pseudo-residuals* of a loss function L. A pseudo-residual is the gradient of that loss function with respect to the current model parameters. This is not a traditional gradient descent method: it is considered *functional gradient descent* [Dua+19]. Friedman [Fri02] later made improvements to gradient boosting framework. These include *bagging* [Bre96] i.e observation sub-sampling and regularization. Nowadays, *feature sub-sampling*, as in Random Forests [Bre01], is also employed. This framework as a whole can be considered to constitute the modern *Gradient Boosting Machine*. This paper focuses exclusively on Gradient Boosting Machines where a decision tree is used as the base learner.

Gradient Boosting Machines approximate the function F^*, that maps $\mathbf{x}$ to y, by

$$F(\mathbf{x}) = \sum_{m=0}^{M} \beta_m h(\mathbf{x}; \mathbf{a}_m) \tag{1}$$

Where functions $h(\mathbf{x}; a_m)$ are the weak learners, which are in this context *Classification and Regression Trees* [Bre+84], or CARTs. They are functions of $\mathbf{x}$ and parametrized by $\mathbf{a}$, which in the case equates to the splitting variables, split locations and terminal node means. Parameters $\mathbf{a}$ and coefficients β are fit to the training data jointly in a stage-wise process. $F(\mathbf{x})$ is fit to minimize the expected loss over the joint distribution of y and $\mathbf{x}$.

$$\{\beta_m, \mathbf{a}_m\} = \arg\min_{\beta, \mathbf{a}} \sum_{i=1}^{n} L(y_i, F_{m-1}(\mathbf{x}) + \beta_m h(\mathbf{x}; \mathbf{a}_m)) \tag{2}$$

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \beta_m h(\mathbf{x}; \mathbf{a}_m) \tag{3}$$

First, $\mathbf{a}$ is obtained by fitting the current weak learner to the pesudo-residuals (negative gradients) of that iteration:

$$\tilde{g}_i = -\left[\frac{\partial L(y_i, F_{m-1}(\mathbf{x}_i))}{\partial F_{m-1}(\mathbf{x}_i)}\right] \tag{4}$$

Next, β_m is obtained by the point at which the loss L in the training data, with parameters $\mathbf{a}$, is the smallest. When using trees as weak learners, a shrinkage parameter (*learning rate*) ν is included in the second term of Eq. (3) to control the rate of learning. It controls how much the newly added tree influences the final additive model. It can be considered a form of regularization that prevents overfitting.

As [Fri01] notes, the depth of the trees, or alternatively the number of terminal nodes, can be used to control the amount of interactions "impilicitly" modeled by the trees. This reduces the need for the practitioners to manually find interactions between features. However this is only true to some extent as it is generally known that domain expertise can help you create superior features, which are technically interactions. In addition to that, the number of trees (*boosting iterations*) M, is another *hyperparameter* to be chosen heuristically by the practitioner or optimized through a hyperparameter optimization procedure. Another advantage of Gradient Boosting Machines is that they decreases the need for manual feature selection, another labour-intensive task in Data Science. This is because trees select relevant features internally and are also robust against inclusion of irrelevant variables [Fri01].

2 Gradient Boosting Machine Implementations

2.1 XGBoost

Extreme Gradient Boosting [CG16], or XGBoost, is a common Gradient Boosting Machine implementation among Data Science practitioners. It is known for scalability, robustness and ability to deal with sparse data. It is intended for data sets with millions of observations and hundreds of features. It may be the most widely used open source, end-to-end tree boosting system [CG16]. It implements gradient tree boosting, as described by Friedman [Fri01]. Its innovations include a sparsity-aware algorithm and a weighted quantile sketch procedure, which enables handling instance wights in approximate tree learning. Also, it exploits out-of-core computation to enable scalability.

XGBoost utilizes a new, regularized learning objective [CG16], which is defined as follows:

$$\begin{aligned} \mathcal{L}_m =& \sum_{i=1}^{n} L(y_i, F_{m-1}(\mathbf{x}_i) + \beta_m h(\mathbf{x}_i)) + \Omega(h_m) \\ \text{where } \Omega(h) =& \gamma T + \frac{1}{2}\lambda \|\mathbf{w}\|^2 \end{aligned} \tag{5}$$

T is the number of leaves and $\mathbf{w}$ the leaf weights of the regression tree. Ω penalizes the complexity of the model (the additive tree functions) by smoothing the final weights of the model and thus reducing overfitting. For computational efficiency, a second-order approximation of the objective is used

$$\begin{aligned} \mathcal{L}_m =& \sum_{i=1}^{n} \left(g_i h_m(\mathbf{x}_i) + \frac{1}{2} g_i' h_m^2(\mathbf{x}_i) \right) + \Omega(h_m) \\ \text{where } g_i =& \frac{\partial L(y_i, F_{m-1}(\mathbf{x}_i))}{\partial F_{m-1}(\mathbf{x}_i)} \\ \text{and } g_i' =& \frac{\partial^2 L(y_i, F_{m-1}(\mathbf{x}_i))}{\partial^2 F_{m-1}(\mathbf{x}_i)} \end{aligned} \tag{6}$$

The gradients g_i are calculated for each observation per iteration. If we define I_j as the set of observations on leaf j, we can rewrite the approximation as follows

$$\mathcal{L}_m = \sum_{j=1}^{T} \left((\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} g_i' + \lambda) w_j^2 \right) + \gamma T \tag{7}$$

For a fixed tree, we can compute the optimal leaf weight w_j^* by

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} g_i' + \lambda} \tag{8}$$

which corresponds to the minimal loss value $\mathcal{L}^*$

$$\mathcal{L}_m^* = -\frac{1}{2}\sum_{j=1}^{T}\frac{(\sum_{I_j} g_i)^2}{\sum_{I_j} g_i' + \lambda} + \gamma T \tag{9}$$

Eq. (9) is used as an *impurity score* for the evaluation of the trees. The loss reduction, or *gain*, after a split is used to choose split candidates. It is given by

$$\mathcal{L}_{split} = \frac{1}{2}\left[\frac{(\sum_{I_L} g_i)^2}{\sum_{I_L} g_i' + \lambda} + \frac{(\sum_{I_R} g_i)^2}{\sum_{I_R} g_i' + \lambda} - \frac{(\sum_{I} g_i)^2}{\sum_{I} g_i' + \lambda}\right] - \gamma \tag{10}$$

Where $I = I_L \cup I_R$ are the observation sets of left and right nodes after the split.

In finding the splits, XGBoost offers two different options for the algorithm [CG16]. The *exact algorithm* iterates over all the possible splitting points greedily. This requires sorting the data first. The *approximate algorithm* proposes splitting points by finding percentiles of the feature distributions. Thus, continuous features are discretized into buckets, which also define the splitting points. Many existing algorithms already utilize this method. However, XGBoost presents a new technique, the *weighted quantile sketch*, to be used with the approximate algorithm. It can handle weighted datasets, ie. datasets where each observation does not have the same weight in terms of contributing to the loss. This would be the case in highly unbalanced classification tasks.

Lastly, XGBoost finds splits in sparse data by adding a *default direction* in each tree node [CG16]. That default direction is used when classifying missing values and it is learned from the data. This sparsity-awareness improves the predictive performance significantly.

The other improvements not discussed in this paper are cache access patterns, data compression and sharding.

XGBoost has gained huge popularity among Data Science practitioners and is known for excellent predictive performance and relatively good scalability. Also, XGBoost has proved to produce results almost up to par with ensemble models, where many different types of models are *stacked* [CG16]. What this implies for a practitioner is convenience as he or she does not have to preprocess and fit multiple models. It also reduces complexity and the possibility parts of the code stop working as intended.

2.2 LightGBM

The Light Gradient Boosting Machine, or LightGBM, was released soon after XGBoost [Ke+17]. It claims to improve efficiency and scalability with high-dimensional data and proposes two new techniques to solve those problems: *Gradient-based One-side Sampling* (GOSS) and *Exclusive Feature Bundling* (EFB).

GOSS is based on the idea that you can improve computational efficiency without reducing predictive performance by excluding observations with small gradients and only using those with large gradients [Ke+17]. Observations with large gradients are considered under-trained (have large errors) and contribute more to the information gain. Therefore, it approximates the gain of a split with less data. In practice, LightGBM keeps observations with large gradients and randomly samples observations with small gradients.

In EFB, mutually exclusive features are combined in order to reduce the dimensionality of the data [Ke+17]. Mutually exclusive features are features that "rarely take non-zero values simultaneously". Thereby, it deals with the sparsity of the feature space. The paper presents two new algorithms: one for finding features that should be bundled and another one for constructing the bundles. This is supposed to reduce the amount of computations done on zero feature values.

LightGBM is stated to outperform XGBoost, which is used as a baseline [Ke+17]. Also LightGBM outperforms a generic Gradient Boosting Machine that does not employ GOSS and EFB. GOSS and EFB are concluded to reduce time consumption without reducing accuracy. In the Data Science community, despite gaining significant popularity and adoption upon its release, LightGBM is widely considered to be faster but not having as good a predictive performance as XGBoost.

2.3 NGBoost

The recently published Natural Gradient Boosting (NGBoost) [Dua+19] differs from the two previous methods in that it uses the *natural gradient* to output a full probability distribution $P(y|\mathbf{x})$ instead of a point estimate $E(y|\mathbf{x})$. In this way, NGBoost quantifies the uncertainty related to the prediction, which is required in fields such as healthcare. Generally, this is called *probabilistic prediction*. However, this approach requires simultaneous boosting of multiple parameters in the Gradient Boosting Machine framework, which is precisely what NGBoost seeks to address.

The algorithm requires three components: a base learner f, a parametric probability distribution P_θ and a scoring rule $\mathcal{S}$ [Dua+19]. The distribution parameters θ are obtained from the gradient boosting process. In the case of a Gaussian, they would be $\theta = (\mu, \sigma)$. This requires one base learner per parameter for each boosting iteration, ie. $f_m = (f_m^\mu, f_m^\sigma)$.

$$\theta = \theta_{m-1} - \nu \sum_{m=1}^{M} \rho_m f_m(\mathbf{x}) \tag{11}$$

Predicted outputs are scaled by ρ_m and the learning rate ν. The natural gradient g_i is calculated for each observation per iteration. It is computed with respect to score $\mathcal{S}$ with parameters θ_{m-1}. The corresponding base learners are fit to those gradients.

The *scoring rule* $\mathcal{S}(P, y)$ is a function that maps a distribution P and obervation y to a score [Dua+19]. A common scoring rule is the log-score (MLE):

$$\mathcal{L}(\theta, y) = -\log P_\theta(y) \tag{12}$$

The natural gradient g is the direction of steepest descent in Riemannian space and is in the case of MLE of form:

$$\tilde{\nabla}\mathcal{S}(\theta, y) \;\propto\; \mathcal{I}(\theta)^{-1}\nabla\mathcal{L}(\theta, y) \tag{13}$$

where $\mathcal{I}(\theta)$ is the Fisher Information

$$\mathcal{I}(\theta) = \mathbb{E}_{y\sim P_\theta}\left[-\nabla_\theta^2\,\mathcal{L}(\theta, y)\right] \tag{14}$$

NGBoost offers state-of-the-art performance and interpretable results, especially in smaller datasets.

3 Summary

When considering predictive regression and classification problems on tabular data, Gradient Boosting Machines have effectively gained the position of an universal approximator among Data Science practitioners. They are end-to-end ensemble models that require little preprocessing, scale well and are relatively robust to outliers and missing data. First, this paper introduced boosting and gradient boosting. Then, it presented three different implementations of the Gradient Boosting Machine: LightGBM, XGBoost and NGBoost. The first two proposed novel techniques to deal with issues

such as high dimensionality, sparsity and high data count while the third one aimed to produce probabilistic predictions.

The boosting algorithm AdaBoost aims to maintain a set of weights over the observations in the data. Weak learners are fit to the data adjusted by the current weights. On each iteration, the error of the newly added learner is calculated and the weights are adjusted to minimize it. The weights are increased for observations misclassified by the current weak learner and decreased for the correctly classified observations. The resulting learning algorithm produces predictions by a weighted, majority vote of the weak learners.

Actual gradient boosting algorithms also work by fitting simple, additive functions. However, the weak learners are iteratively fit to the pseudo-residuals of the loss function. Pseudo-residuals are the gradient of that loss function with respect to the current model parameters. Later improvements to the gradient boosting framework include observation sub-sampling, regularization and feature sub-sampling. This framework as a whole constitutes the modern Gradient Boosting Machine. Gradient Boosting Machines reduce the need for manual feature engineering and feature selection.

XGBoost may be the most widely used open-source tree boosting system. It is known for scalability, robustness and ability to deal with sparse data. Its innovations include a sparsity-aware algorithm and an improvement to approximate tree learning: the weighted quantile sketch procedure. Also, it introduces an addition to the learning objective, a regularization term. Other augmentations not discussed in this paper are cache access pattern, data compression and sharding.

LightGBM claims to improve efficiency and scalability with high-dimensional data and proposes two new techniques to solve those problems: Gradient-based One-side Sampling (GOSS) and Exclusive Feature Bundling (EFB). In GOSS, only observations with large gradients are used and observations

with small gradients are sampled randomly. In EFB, mutually exclusive features are combined to reduce the dimensionality of the data and deal with sparsity. GOSS and EFB are concluded to reduce time consumption without reducing accuracy.

The recently published NGBoost differs from the two previous methods in that it outputs a full probability distribution instead of a point estimate, thus quantifying the uncertainty related to the prediction. The algorithm requires three components: a base learner, a parametric probability distribution, and a scoring rule. As previously, the parameters are obtained from the gradient boosting process but this time the base learners are fit to the natural gradients instead of pseudo-residuals.

References

[Bre+84] L. Breiman et al. *Classification and Regression Trees*. The Wadsworth and Brooks-Cole statistics-probability series. Taylor & Francis, 1984. ISBN: 9780412048418.

[Bre01] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.

[Bre96] Leo Breiman. "Bagging predictors". In: *Machine learning* 24.2 (1996), pp. 123–140.

[CG16] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16* (2016).

[Dua+19] Tony Duan et al. "NGBoost: Natural Gradient Boosting for Probabilistic Prediction". In: *arXiv preprint arXiv:1910.03225* (2019).

[Fri01] Jerome H. Friedman. "Greedy Function Approximation: A Gradient Boosting Machine". In: *The Annals of Statistics* 29.5 (2001), pp. 1189–1232.

[Fri02] Jerome H Friedman. "Stochastic Gradient Boosting". In: *Computational statistics & data analysis* 38.4 (2002), pp. 367–378.

[FS97] Yoav Freund and Robert E Schapire. "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". In: *Journal of Computer and System Sciences* 55.1 (1997), pp. 119–139.

[Ke+17] Guolin Ke et al. "LightGBM: A highly Efficient Gradient Boosting Decision Tree". In: *Advances in neural information processing systems* (2017), pp. 3146–3154.