Uncertainty Estimation for Regression with Gradient Boosting Decision Trees

Sebastian Haglund El Gaidi

sebastian@greenlytics.io

Mars 5, 2020

1 Introduction

Gradient boosting decision trees (GBDTs) is a machine learning algorithm that is part of the family of gradient boosting (or gradient boosting machines). This algorithms has shown strong performance in many classification and regression tasks [1]. The objective of this report is to review differences and advantages relating to regression uncertainty estimation of the most popular gradient boosting implementations: Scikit-Learn, LightGBM, XGBoost and CatBoost.

2 Uncertainty estimation in regression

2.1 Quantile regression (L_1-regression)

Quantile regression refers to fitting a regressor to a quantile loss function (also know as the pinball loss function). The quantile loss function is given by

$$L^q(\hat{y}_\alpha, y) = \alpha(\hat{y}_\alpha - y)_- + (1-\alpha)(\hat{y}_\alpha - y)_+ \tag{1}$$

where $\hat{y}_\alpha$ is the α-quantile prediction of the target variable y. Optimising a regressor $\hat{y}_\alpha = F(x)$ based on some input x for this loss function is an unbiased estimator of the conditional α-quantile denoted $\text{Quantile}_\alpha[Y|X = x]$. For $\alpha = 0.5$ quantile regression is the same as L_1-regression and leads to fitting the conditional median denoted $\text{Median}[Y|X = x]$.

2.2 Expectile regression (L_2-regression)

Expectile regression is done by fitting a regressor to to the expectile loss function given by

$$L^e(\hat{y}_\alpha, y) = \alpha(\hat{y}_\alpha - y)_-^2 + (1-\alpha)(\hat{y}_\alpha - y)_+^2. \tag{2}$$

Optimising a regressor $\hat{y}_\alpha = F(x)$ based on some input x for this loss function is an unbiased estimator of the conditional α-expectile denoted Expectile$_\alpha[Y|X = x]$. For $\alpha = 0.5$ expectile regression is the same as L_2-regression and leads to fitting the conditional mean denoted Mean$[Y|X = x] = \mathbb{E}[Y|X = x]$. Given a set of expectiles, it is possible to numerically obtain the quantile function [2].

2.3 Huberile regression

Another loss that potentially could be used for uncertainty estimation is the Huber loss function defined as

$$L^h(\hat{y}_\alpha, y) = \begin{cases} \frac{1}{2}L^e(\hat{y}_\alpha, y), & |\hat{y}_\alpha - y| \leq \delta \\ \delta L^q(\hat{y}_\alpha, y) - \frac{1}{2}\delta^2(1-\alpha), & (\hat{y}_\alpha - y) > \delta \\ \delta L^q(\hat{y}_\alpha, y) - \frac{1}{2}\delta^2\alpha, & (\hat{y}_\alpha - y) < -\delta. \end{cases} \tag{3}$$

where a α-parameter has been added in analogy with equation (1) and (2). We call the loss function the "Huberile loss function" and the predictions Huberiles. The Huberile loss function tries to maintain the robustness of the quantile loss function and at the same time ensure a continuous derivative.

2.4 Maximum log-likelihood estimation (MLE)

A common way of uncertainty estimation for regression problems is to first parametrise the uncertainty through pre-defined probability distribution $P_\theta(y)$ and then train a regressor to predict the parameters θ by maximising the log-likelihood over the training samples. This is usually refered to as maximum likelihood estimation (MLE) with a loss function given by

$$L^{mle}(\theta, y) = -\log P_\theta(y). \tag{4}$$

The parameters are predicted using a regressor $\hat{\theta} = F_\theta(x)$ and the probability distribution can then be written as $P(y|x) = P_\theta(y) = P(y;\theta) = P(y; F_\theta(x))$.

2.5 Continuous ranked probability score (CRPS)

The continuous ranked probability score (CRPS) is often considered to be a robust alternative to MLE. The loss function for CRPS is given by

$$L^{crps}(\theta, y) = \int_{-\infty}^{y} F_\theta(y)^2 dy + \int_{y}^{\infty} (1 - F_\theta(y))^2 dy. \tag{5}$$

Although being robust, it might not be the simplest loss function to train regressors on.

2.6 Conformal prediction (CP)

Conformal prediction is a non-parametric technique for constructing prediction intervals with valid coverage with finite samples and no assumptions on distributions. In conformal prediction, a regressor is first fitted using samples from a training dataset. Then a test (hold out) dataset is used to construct coverage interval given a confidence level. By the default the coverage interval is not dependent on the the input, but can be extended to allow for heteroscedasticity [3].

2.7 Gradient boosting decision trees

2.7.1 Gradient boosting

Consider the regression task of fitting an approximation $\hat{y} = F(x)$ of some real-valued function $y \in \mathbb{R}$ given a training dataset $\{x_i, y_i\}_{i=1}^{N}$, were $x \in \mathbb{R}^k$ is a vector of predictor variables. Decision trees partition the space of all joint predictors variable values into disjoint regions R_j for $j \in \{1, 2, ..., J\}$. These regions are represented by terminal nodes of the tree called leaf nodes. In every leaf node, a constant γ_j is assigned as the prediction. In gradient boosting decision trees, we seek an approximation $F_M(x)$ in the form of a sum of M decision trees $f_m(x) \in \mathcal{F}$ (weak learners) according to

$$F_M(x) = \sum_{m=0}^{M} f_m(x) = F_0(x) + \sum_{m=1}^{M} \gamma_{jm} \mathbb{1}(x \in R_{jm}) \tag{6}$$

where $f_m(x) = \gamma_{jm} \mathbb{1}(x \in R_{jm})$ with γ_{jm} and R_{jm} being the leaf value and the region of the j^{th} leaf in the m^{th} tree respectively. We find $f(x)$ in an greedily additive manner, so that it minimises the loss function over the training dataset as

$$F_0(x) = \arg\min_{\hat{y}} \sum_{i=1}^{N} L(\hat{y}, y_i) \tag{7}$$

$$F_m(x) = F_{m-1}(x) + \arg\min_{f_m \in \mathcal{F}} \left[\sum_{i=0}^{N} L\Big(F_{m-1}(x_i) + f_m(x_i), y_i\Big) \right] \tag{8}$$

where $F_0(x)$ is a constant initial approximation of y. An approximate solution to equation (8) can be found as

$$F_m(x) = F_{m-1}(x) - \delta_m g_m(x) \tag{9}$$

with

$$g_m(x) = \frac{\partial L\big(F(x), y\big)}{\partial F}\bigg|_{F(x)=F_{m-1}(x)}.$$

Equation (9) can be seen as functional gradient descent with the difference that the gradients $g_m(x)$ are constrained to be the predictions of a J_m-terminal node decision tree structure. The parameter δ_m is the step length that is chosen through line search by solving

$$\delta_m = \arg\min_{\gamma} \sum_{i=0}^{N} L\Big(F_{m-1}(x_i) - \delta g_m(x_i), y_i\Big). \tag{10}$$

2.7.2 Calculating the step and loss given an additive tree

In order to solve equation (10), we need to find a decision tree $f_m(x) = \gamma_{jm}\mathbb{1}(x \in R_{jm})$ that approximates the negative gradients of the previous tree iteration as

$$(\gamma_{jm}, R_{jm}) = \arg\min_{\gamma, R} \sum_{i=0}^{N} L_s\Big[- g_m(x_i), \gamma\mathbb{1}(x_i \in R)\Big] \tag{11}$$

where L_s is a surrogate loss usually taken as the squared loss i.e. $L_s(a, b) = (a - b)^2$. Estimating γ_{jm} is typically trivial when the regions R_{jm} of the tree structure are determined. Considering the surrogate loss to be the squared loss then

$$\begin{aligned} L(f_m) &= \sum_{i=1}^{N} \Big(- g_m(x_i) - f_m(x_i)\Big)^2 \\ &= \sum_{i=1}^{N} \Big(g_m(x_i)^2 + 2g_m(x_i)f_m(x_i) + f_m(x_i)^2\Big) \\ &= \sum_{j=1}^{J_m} \Big(\sum_{i \in R_{jm}} g_m(x_i)^2 + 2\gamma_{jm} \sum_{i \in R_{jm}} g_m(x_i) + n_{jm}\gamma_{jm}^2\Big) \end{aligned} \tag{12}$$

where n_{jm} is the number of samples in region R_{jm}. Minimising the loss function with respect to the leaf values we get

$$\gamma_{jm} = -\frac{\sum_{i \in R_{jm}} g_m(x_i)}{n_{jm}} = -\frac{G_{jm}}{n_{jm}} \tag{13}$$

where G_{jm} is the sum of the gradients in region R_{jm}. This means, given the tree structure, we can easily calculate the leaf values as the mean of the negative gradients. Inserting the leaf values back into the surrogate loss yields

$$L(f_m) = \sum_{j=1}^{J_m} \Big(- \frac{G_{jm}^2}{n_{jm}} + \sum_{i \in R_{jm}} g_m(x_i)^2\Big). \tag{14}$$

Instead of using a first-order approximation of the loss function, as the case with gradient descent, a second-order approximation can be used as

$$
\begin{aligned}
L(f_m) &= \sum_{i=1}^{N} L\Big(F_{m-1}(x_i) + f_m(x_i), y_i\Big) \\
&\approx \sum_{i=1}^{N} L\Big(F_{m-1}(x_i), y_i\Big) + g_m(x_i) f_m(x_i) + \frac{1}{2} h_m(x_i) f_m(x_i)^2 \\
&= \sum_{j=1}^{J_m} \sum_{i \in R_{jm}} L\Big(F_{m-1}(x_i), y_i\Big) + g_m(x_i)\gamma_{jm} + \frac{1}{2} h_m(x_i)\gamma_{jm}^2 \\
&\propto \sum_{j=1}^{J_m} \sum_{i \in R_{jm}} g_m(x_i)\gamma_{jm} + \frac{1}{2} h_m(x_i)\gamma_{jm}^2 \qquad (15)
\end{aligned}
$$

with

$$
h_m(x) = \frac{\partial^2 L\big(F(x), y\big)}{\partial F^2}\bigg|_{F(x)=F_{m-1}(x)}.
$$

being the hessian of the loss function evaluated at $F_{m-1}(x)$. For a given tree structure, it is easy to find the optimal leaf values to be

$$
\gamma_{jm} = -\frac{\sum_{i \in R_{jm}} g_m(x_i)}{\sum_{i \in R_{jm}} h_m(x_i)} = -\frac{G_{jm}}{H_{jm}} \qquad (16)
$$

where H_{jm} is the sum of the hessian in the region R_{jm}. Plugging it back into the second-order approximation of the loss function gives

$$
L(f_m) \propto -\frac{1}{2} \sum_{j=1}^{J_m} \frac{G_{jm}^2}{H_{jm}} \qquad (17)
$$

2.7.3 Calculating the splitting point

The splitting points of $f_m(x)$ are determined from minimising the loss $L(f_m)$ given by

$$
L(f_m) = \sum_{i=1}^{N} \sum_{j=1}^{J_m} L\Big(y_i, \gamma_{jm} \mathbb{1}(x_i \in R_{jm})\Big) \equiv \sum_{j=1}^{J_m} L_j. \qquad (18)
$$

Now, consider splitting region R_k into two new regions R_{kL} and R_{kR} (so that $R_k = R_{kL} \cup R_{kR}$). The loss is then given by

$$
\begin{aligned}
L(f_{before}) &= L_k + \sum_{j \neq k} L_j \\
L(f_{after}) &= L_{kL} + L_{kR} + \sum_{j \neq k} L_j
\end{aligned}
$$

and we can calculate the improvement of the loss (gain) as

$$I_{gain} = L(f_{before}) - L(f_{after}) = L_k - (L_{kL} + L_{kR}). \tag{19}$$

Substituting in the first-order loss from equation (14) into (19) gives

$$\begin{aligned} I_{gain} =& -\frac{G_{km}^2}{n_{km}} + \sum_{i \in R_{km}} g_m(x_i)^2 - \\ & \Big(-\frac{G_{kLm}^2}{n_{kLm}} + \sum_{i \in R_{kLm}} g_m(x_i)^2 - \frac{G_{kRm}^2}{n_{kRm}} + \sum_{i \in R_{kRm}} g_m(x_i)^2 \Big) \\ =& \frac{G_{kLm}^2}{n_{kLm}} + \frac{G_{kRm}^2}{n_{kRm}} - \frac{G_{km}^2}{n_{km}}. \end{aligned} \tag{20}$$

Substituting with the second-order loss from equation (15) into (19) instead gives

$$I_{gain} = \frac{1}{2}\Big(\frac{G_{kLm}^2}{H_{kLm}} + \frac{G_{kRm}^2}{H_{kRm}} - \frac{G_{km}^2}{H_{km}} \Big). \tag{21}$$

Notice that equations (20) and (21) allow to speed up calculations significantly since the loss function does not need to be calculated for every splitting point for all features. Instead, gradients and hessians are summed by sweeping only once over each feature. In order to avoid overfitting, it is common to add L1- and L2 regularisation to the loss of equation (15), which yields

$$L(f_m) = \sum_{j=1}^{J_m} \Big(G_{jm}\gamma_{jm} + \frac{1}{2} H_{jm}\gamma_{jm}^2 \Big) + \frac{1}{2}\lambda \sum_{j=1}^{J_m} \gamma_{jm}^2 + \alpha \sum_{j=1}^{J_m} |\gamma_{jm}|. \tag{22}$$

From equation (22) the optimal leaf values with regularisation becomes

$$\gamma_{jm} = \begin{cases} -\frac{G_{jm}+\alpha}{H_{jm}+\lambda}, & G_{jm} < -\alpha \\ -\frac{G_{jm}-\alpha}{H_{jm}+\lambda}, & G_{jm} > \alpha \\ 0, & \text{else} \end{cases} \tag{23}$$

effectively setting $\gamma_{jm} = 0$ when $|G_{jm}| < \alpha$ and the gain becomes

$$I_{gain} = \begin{cases} -\frac{1}{2}\Big(\frac{(G_{kLm}+^2}{H_{kLm}} + \frac{(G_{kRm}+\alpha)^2}{H_{kRm}} - \frac{(G_{km}+\alpha)^2}{H_{km}} \Big), & G_{jm} < -\alpha \\ -\frac{1}{2}\Big(\frac{(G_{kLm}-^2}{H_{kLm}} + \frac{(G_{kRm}-\alpha)^2}{H_{kRm}} - \frac{(G_{km}-\alpha)^2}{H_{km}} \Big), & G_{jm} > \alpha \\ 0, & \text{else.} \end{cases} \tag{24}$$

3 GBDT implementations

3.1 Scikit-Learn Implementation

Gradient boosting in the Scikit-Learn implementation uses gradient descent to find the additive tree structure [4]. The implementation follows the method developed by Friedman [5]. First the negative gradients of the loss function are calculated. Then the additive tree structure is built by fitting to the negative gradients using a surrogate loss. When the true loss function and the surrogate loss are both the mean squared loss, this procedure falls back to be equivalent with least-square boosting and the leaf values are given by the average of the gradients according to equation (14). Another surrogate loss implemented in Scikit-Learn is mean absolute error and "Friedman mean squared error" which is generally the best as it can provide a better approximation in some cases [**?**]. The final predictions in each leaf are calculated based on the true loss function and not the surrogate loss function.

3.2 XGBoost Implementation

XGBoost is a scalable end-to-end tree boosting system that achieves state-of-the-art results in many machine learning competitions [1]. It implements a sparsity-aware algorithm for sparse data and weighted quantile sketch for approximate tree learning. The sparsity-aware algorithm classifies samples in the optimal default direction that is learn from the data. The weighted quantile sketch approximation refers to binning continuous features and only considering split points at the bins. If bins are calculated locally (at every tree level) then the performance reduction compared to the greedy version of using all split points remains small.

3.3 LightGBM Implementation

LightGBM is a highly efficient implementation of GBDT in terms of computational speed and memory consumption developed by Microsoft Research [6]. It introduces two novel techniques: Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB). The rationale behind the GOSS technique is that samples with larger gradients will contribute more to the information gain (as can be seen from equation (21)). The data instances with small gradients are randomly dropping limiting the need to scan through all samples to calculate the information gain. In EFB features that are (almost) mutually exclusive are bundled. Here, mutually exclusiveness refers to rarely taking nonzero values simultaneously. Finding the best splitting points in the decision trees is the most computational expensive part of GBDT. By using GOSS and EFB significantly reduces those need for computations while achieving almost the same accuracy as standard GBDT. In addition to these two techniques, LightGBM relies on a histogram-based algorithm to find split points, which also speeds up computations.

3.4 Catboost Implementation

CatBoost is a GBDT implementation that is specialised on handling categorical features [7]. Categorical features cannot be used in binary decision trees directly. It is therefore common to preprocess categorical features into numerical ones. Typically, for categorical features of low cardinality this is done using one-hot encodings. Another approach to deal with categorical features is to calculate some statistics based on the label values. CatBoost implements a statistic that is based on permutations of the dataset in order to reduce overfitting. In addition, CatBoost tries to reduce the problem of gradient bias from using gradients for both splitting and leaf values in standard GBDT implementations. Gradient for splitting the tree is estimated using samples that were not used to construct the base model (tree ensemble) of which the gradients are based on. In CatBoost, trees are grown using a level-wise strategy.

3.5 Available objective functions

Different GBDT implementations support different objective functions out of the box. Below is a summary of the supported objective functions from Scikit-learn, XGBoost, LightGBM and CatBoost.

Table 1: Supported objective functions for GBDT implementations.

	Quantile		Expectile		Huberile		NLL	CRPS	CP
Regression-α	0.5	Any	0.5	Any	0.5	Any			
Scikit-learn	Yes	Yes	Yes	No	Yes	Yes	No	No	No
XGBoost	No	No	Yes	No	No	No	No	No	No
LightGBM	Yes	Yes	Yes	No	No	No	No	No	No
CatBoost	Yes	Yes	Yes	Yes	Yes	No	No	No	No

4 Quantile regression in LightGBM and CatBoost

Quantile regression in LightGBM is implemented in two steps [8]. First the decision tree structure is built optimising for the gain in equation (21). Since hessians are zero for the quantile loss function, these are set to unity, effectively imposing and L2 regularisation with $\lambda = 1$ in order to avoid a zero denominator. Then the quantile leaf values are recalculated by optimising the true (quantile) loss function over the data samples contained in the leaf. This avoids using corrupted leaf values as the zero hessian in equation (23) could not provide locational information. CatBoost has implemented quantile regression in a similar manner as LightGBM. First, a tree is grown using gradients and then all leaf values are updated using the true loss function. The local optimisation of the quantile loss function amounts to finding the empirical quantile of the residuals.

In order to achieve good convergence for the quantile loss function, it is essential that the constant base function $F_0(x)$ in equation (8) is set in a appropriate way. If all samples give rise to only negative (or positive) residuals then, since the gradients are $\text{sign}(F(x_i) - y_i)$, all gradients will have the same value (either -1 or 1) and learning will be hampered. To avoid this, the base function is set to the quantile of the data distribution (all labels) that is optimised for.

5 Experiments

5.1 Experiment setup

Several experiments are conducted using the considered implementations of GBDT. The experiements are performed by training the GBDTs on a synthetic dataset and then evaluating the quality of quantile predictions visually and quantitatively. The synthetic dataset is given by

$$f(x) = c + x\sin(x) + A \tag{25}$$

where $\varepsilon_{noise} \sim \mathcal{N}(0, B)$ with $B \sim 1.5 + \mathcal{U}(0, 1)$. In all experiments the number of sampled data points is $N = 10000$ and the quantiles to be estimated are $\alpha = \{0.1, 0.5, 0.9\}$.

5.2 Experiment analysis

The Scikit-Learn implementation can be seen in the experiments to provide robust quantiles even when the starting point for the boosting procedure is set to zero. The main reason for this the real loss is used to calculate the leaf values. In the LightGBM and CatBoost implementations, quantile predictions provide high quality as long as the true loss is used to calculate the leaf values and enough regression trees are used in the ensemble. When the gradient strategy is used to update the leaf values it can be seen that quality and convergence is highly dependent on the starting point of boosting as well as the amount of regression trees. The XGBoost implementation does not have any option to calculate leaf values using the true loss function instead of gradients and therefore suffers the same quality and convergence issues as LightGBM and CatBoost in all experiments.

6 Discussion

Different implementations of GBDTs have different advantages and disadvantages. For the task of uncertainty estimation it is clear that it is possible to for all implementations to perform quantile regression (dispite zero hessians) by updated the leaf values using the true loss function. There exist both computational and quality benefits of expectile regression as compared to quantile regression. However, expectiles cannot be intuitively understood as quantiles can.

In the conducted experiments, a separate model was trained for each quantile. When many quantiles are to be estimated this might result in a cumbersome procedure. In addition, there is no guarantee of non-crossing quantiles when training separate models. In principle it should be possible to develop a GBDT to provide all quantiles from a single regressor. First, the loss of all quantiles would be average in the true loss function used for building the tree structure. Next all quantiles from each leaf node would be outputted as the prediction instead of only one quantile. This would guarantee that the quantiles are non-crossing, however it would reduce the ability of the quantile prediction models to specialise on certain features.

There is some work on implementation of other ways of providing uncertainty estimates using GBDTs. These include predicting parameters of pre-defined probability distributions and optimising fit using maximum likelihood estimation (MLE) or continuous rank probability score (CRPS) [9, 10]. Conformal prediction (CP) can be used to provide statistical guarantees regarding coverage of any type of regressor. Some work has been done to combine conformal prediction with quantile regression in order to be able to model heteroskedasticity in the data [11]. However, most of this work is still in an early stage an no robust open-source implementations exist.

7 Acknowledgements

This report is part of the project "Optimal Wind Power Trading Probabilistic Forecasting." funded by the Swedish Energy Agency within the VindEL program project number 47070-1.

References

[1] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. 2016.

[2] L. Schulze Waltrup, F. Sobotka, T. Kneib, and G. Kauermann. Expectile and quantile regression—david and goliath? 2015.

[3] U. Johansson, H. Boström, T. Tuve Löfström, and H. Linussons. Regression conformal prediction with random forests. 2015.

[4] Scikit-Learn. Scikit-learn 1.11. ensemble methods. URL: https://scikit-learn.org/stable/modules/ensemble.htmlgradient-tree-boosting.

[5] J. H. Friedman. Greedy function approximation: A gradient boosting machine. 2001.

[6] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, Ma W., Q. Ye, and T. Liu. Lightgbm: A highly efficient gradient boosting decision tree. 2017.

[7] A. V. Dorogush, V. Ershov, and Gulin A. Catboost: gradient boosting with categorical features support. 2017. Yantex.

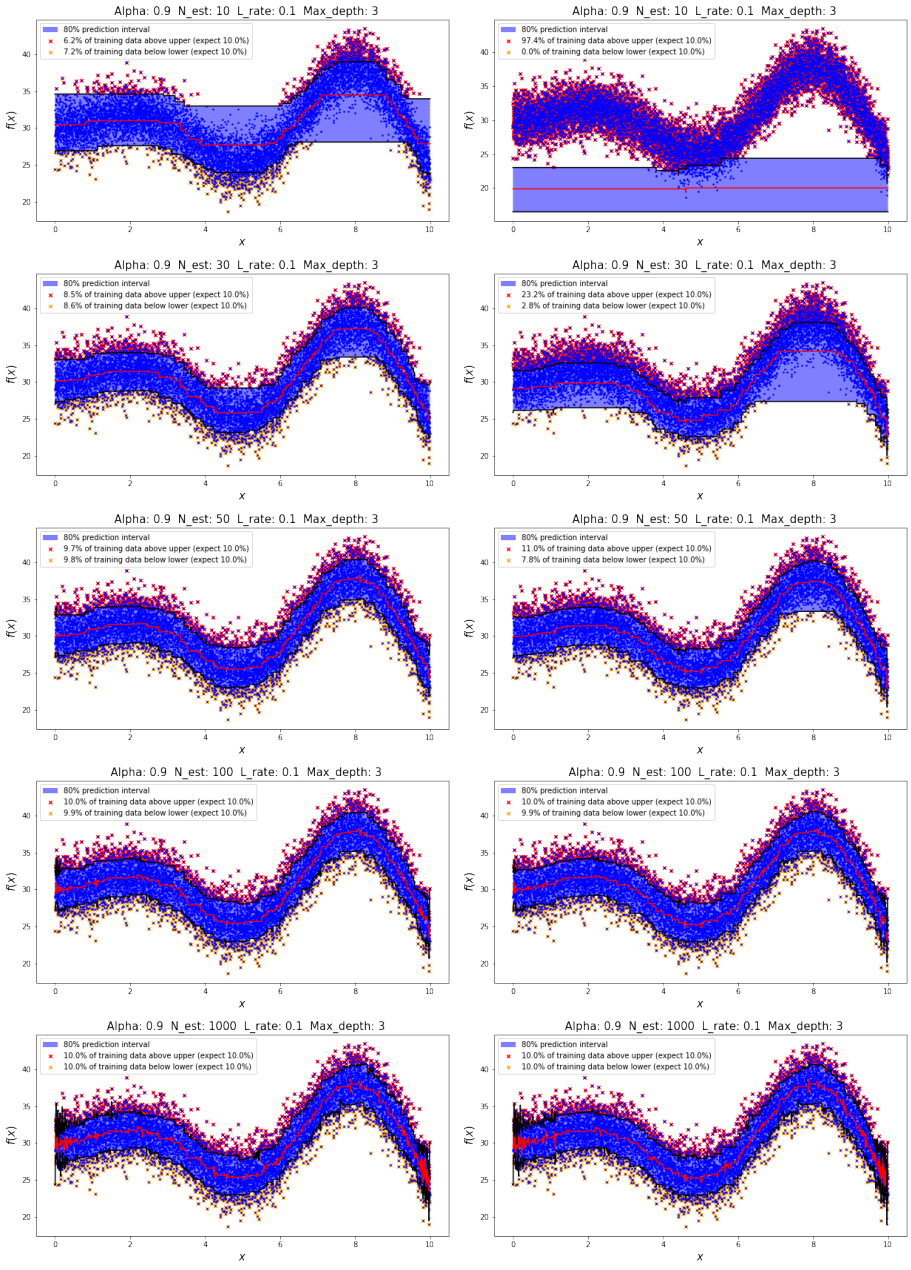
[8] M.J. Hou. How lightgbm implements quantile regression. 2018. URL: http://jmarkhou.com/lgbqr/.

[9] T. Duan, A. Avati, D. Yi Ding, K. K. Thai, S. Basu, A. Ng, and A. Schuler. Ngboost: Natural gradient boosting for probabilistic prediction. 2020.

[10] Alexander Marz. Xgboostlss - an extension of xgboost to probabilistic forecasting. 2020.

[11] Y. Romano, E. Patterson, and E. J. Candes. Conformalized quantile regression. 2019.

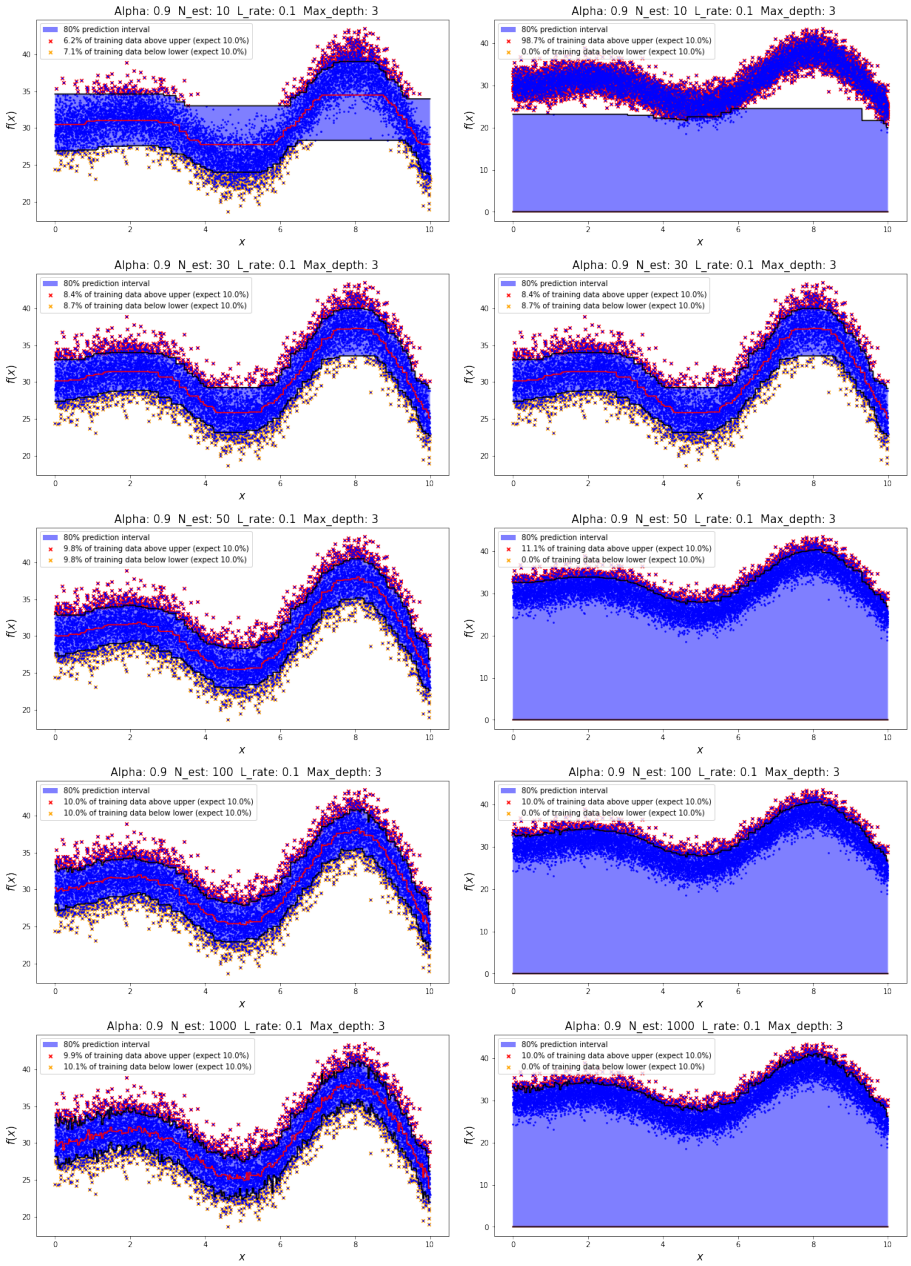
Experiments using Scikit-Learn implementation



(a) Starting at optimal quantile. (b) Starting at zero.

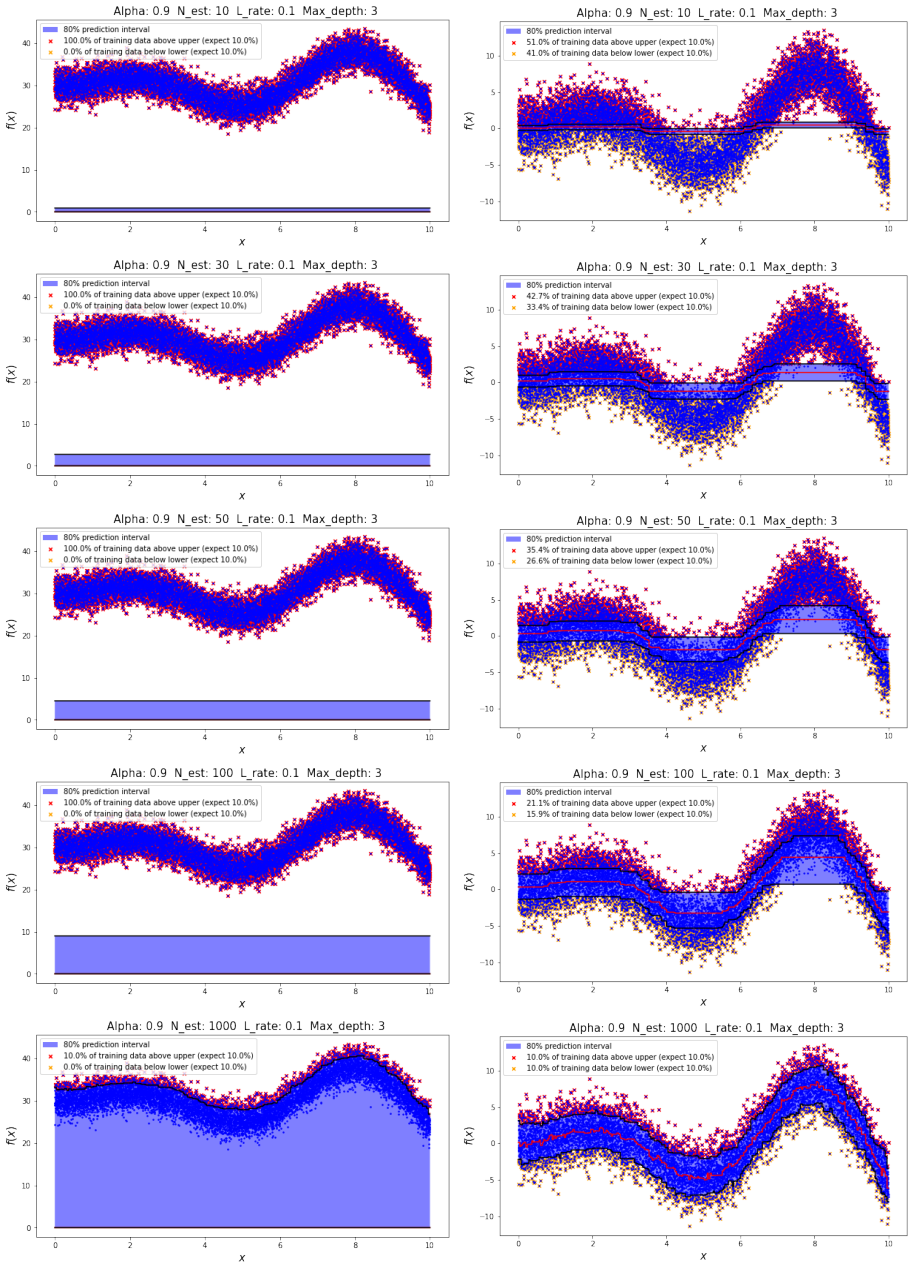
Figure 1: A figure with two subfigures

Experiments using LightGBM implementation



(a) Starting at optimal quantile. (b) Starting at zero.

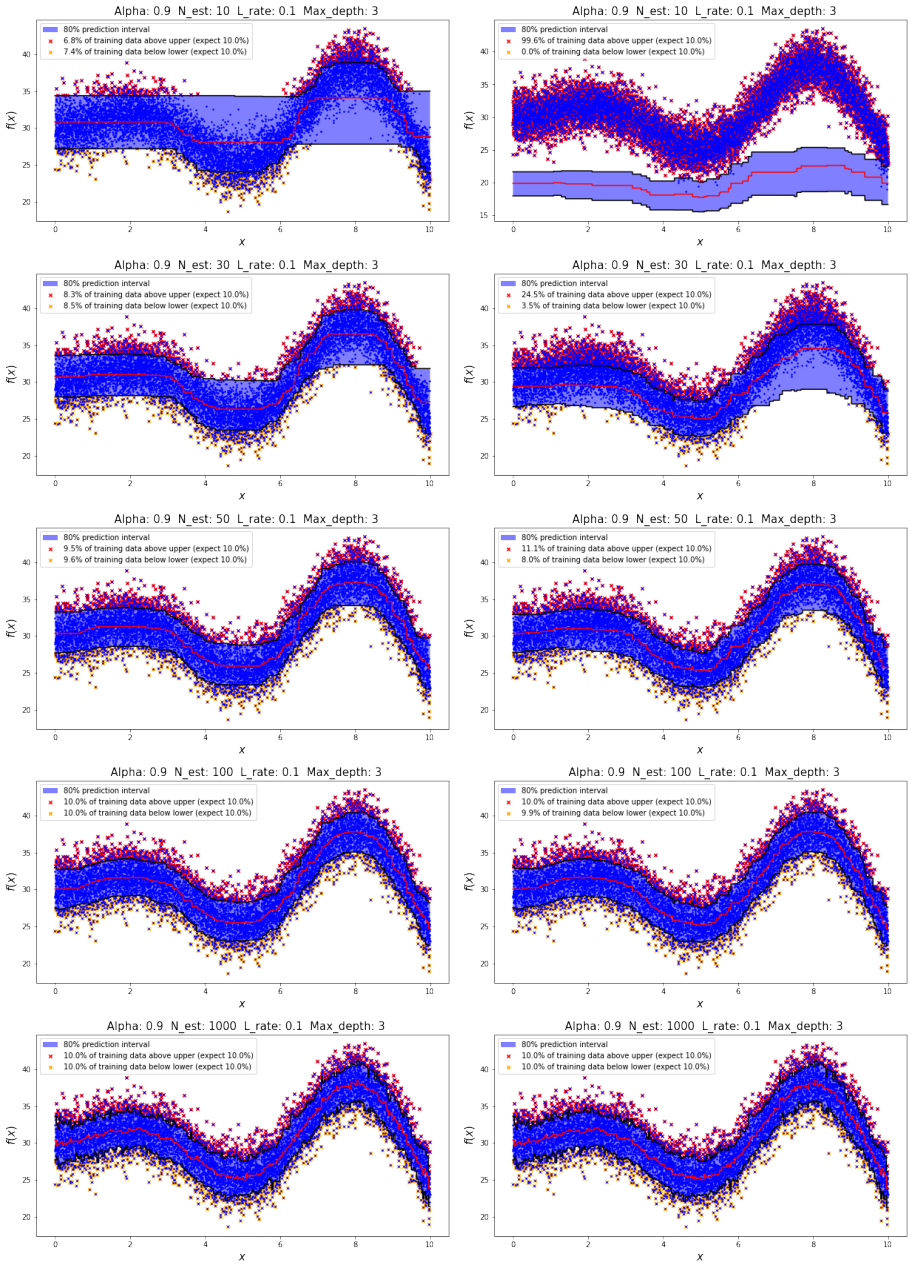
Figure 2: Experiments using LightGBM calculating leaf values optimally.



(a) $a = 30$ (b) $a = 0$

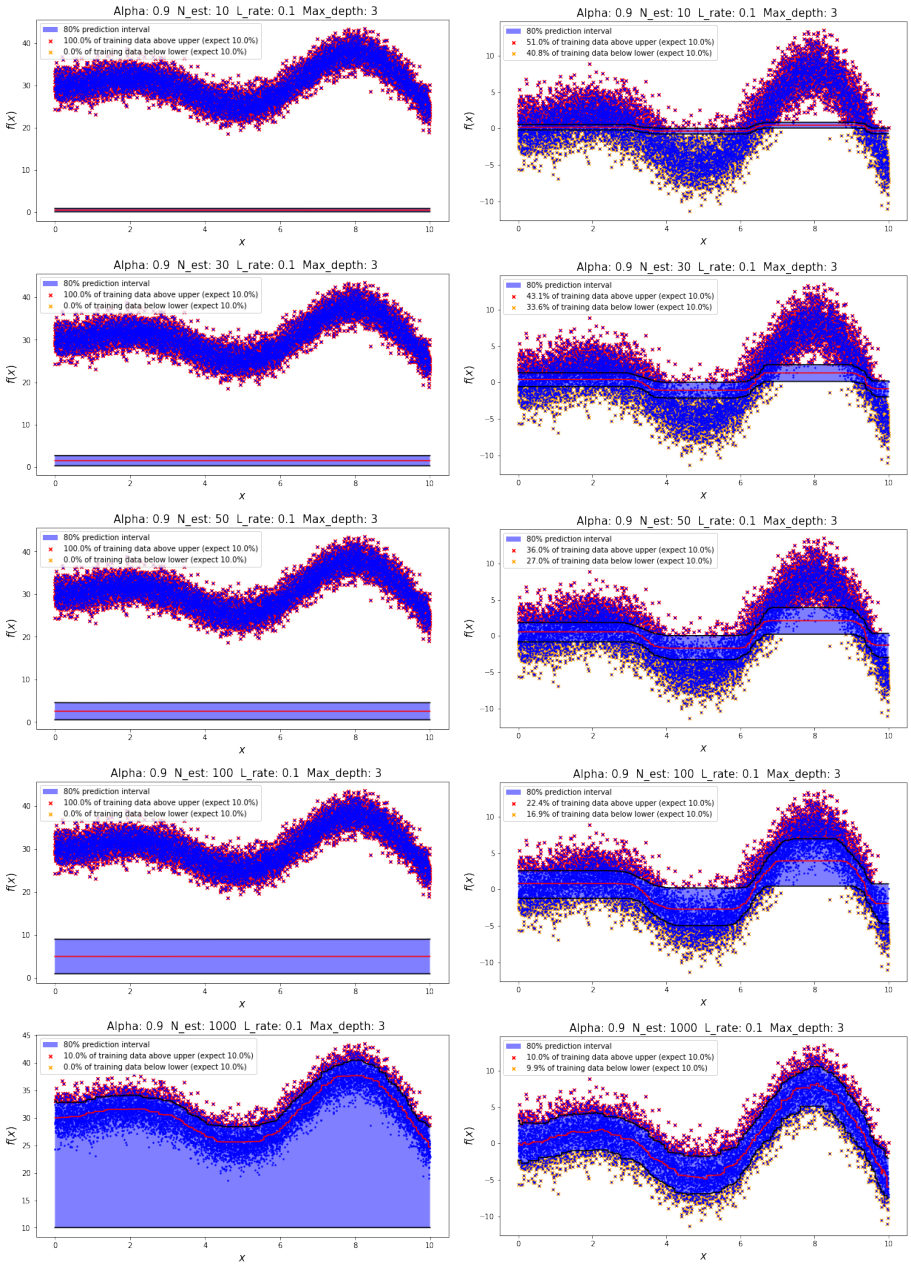
Figure 3: Experiments using LightGBM calculating leaf values with gradients.

Experiments using CatBoost implementation



(a) Starting at optimal quantile. (b) Starting at zero.

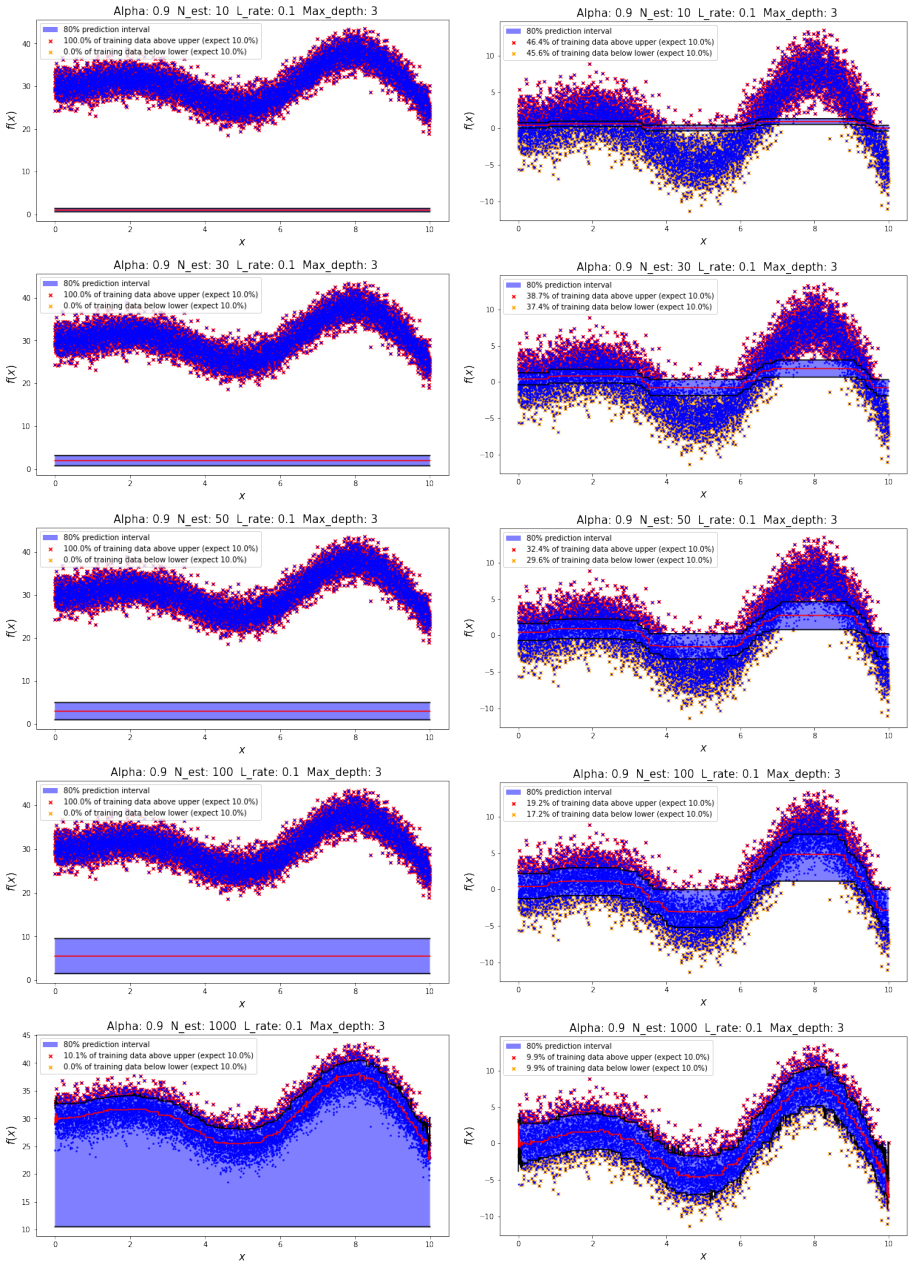
Figure 4: Experiments using CatBoost calculating leaf values optimally.



(a) $a = 30$ (b) $a = 0$

Figure 5: Experiments using CatBoost calculating leaf values using gradients.

Experiments using XGBoost implementation



(a) $a = 30$ (b) $a = 0$

Figure 6: Experiments using XGBoost calculating leaf values using gradients.