NGBoost: Natural Gradient Boosting for Probabilistic Prediction

Chella Thiyagarajan N
ME17B179
EE5180
`Final Term Paper`

Abstract

We present Natural Gradient Boosting (NGBoost), an algorithm for generic probabilistic prediction via gradient boosting. Typical regression models return a point estimate, conditional on covariates, but probabilistic regression models output a full probability distribution over the outcome space, conditional on the covariates. This allows for predictive uncertainty estimation — crucial in applications like healthcare and weather forecasting. NGBoost generalizes gradient boosting to probabilistic regression by treating the parameters of the conditional distribution as targets for a multiparameter boosting algorithm. Furthermore, we show how the Natural Gradient is required to correct the training dynamics of our multiparameter boosting approach. NGBoost can be used with any base learner, any family of distributions with continuous parameters, and any scoring rule

1 Introduction

The reign of the Gradient Boosters were almost complete in the land of tabular data. In most real world as well as competitions, there was hardly a solution which did not have at least one model from one of the gradient boosting algorithms. But as the machine learning community matured, and the machine learning applications started to be more in use, the need for uncertainty output became important. For classification, the output from Gradient Boosting was already in a form which lets you understand the confidence of the model in its prediction. But for regression problems, it wasn't the case. The model spat out a number and told us this was its prediction. How do you get uncertainty estimates from a point prediction? And this problem was not just for Gradient Boosting algorithms. but was for almost all the major ML algorithms. This is the problem that the new kid on the block — NGBoost seeks to tackle.

Estimating the uncertainty in the predictions of a machine learning model is crucial for production deployments in the real world. Not only do we want our models to make accurate predictions, but we also want a correct estimate of uncertainty along with each prediction. When model predictions are part of an automated decision-making workflow or production line, predictive uncertainty estimates are important for determining manual fallback alternatives or for human inspection and intervenion.

Probabilistic prediction (or probabilistic forecasting), which is the approach where the model outputs a full probability distribution over the entire outcome space, is a natural way to quantify those uncertainties.

2 Overview

Gradient Boosting methods have generally been among the top performers in predictive accuracy over structured or tabular input data.

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

NGBoost enables predictive uncertainty estimation with Gradient Boosting through probabilistic predictions (including real valued outputs). With the use of Natural Gradients, NGBoost overcomes technical challenges that make generic probabilistic prediction hard with gradient boosting

The key innovation in NGBoost is the use of Natural Gradients instead of regular gradients in the boosting algorithm. And by adopting this probabilistic route, it models a full probability distribution over the outcome space, conditioned on the covariates.

The paper modularizes their approach into three components -

- Base Learner
- Parametric Distribution
- Scoring Rule

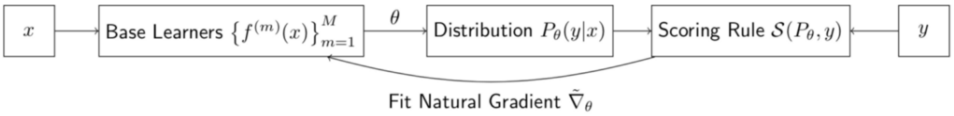


Figure 1: NGBoost Model

2.1 Base Learners

As in any boosting technique, there are base learners which are combined together to get a complete model. And the NGBoost doesn't make any assumptions and states that the base learners can be any simple model. The implementation supports a Decision Tree and ridge Regression as base learners out of the box. But you can replace them with any other sci-kit learn style models just as easily.

2.2 Parametric Distribution

Here, we are not training a model to predict the outcome as a point estimate, instead, we are predicting a full probability distribution. And every distribution is parametrized by a few parameters. For eg, the normal distribution is parametrized by its mean and standard deviation. You don't need anything else to define a normal distribution. So, if we train the model to predict these parameters, instead of the point estimate, we will have a full probability distribution as the prediction.

2.3 Scoring Rule

Any machine learning system works on a learning objective, and more often than not, it is the task of minimizing some loss. In point prediction, the predictions are compared with data with a loss function. Scoring rule is the analogue from the probabilistic regression world. The scoring rule compares the estimated probability distribution with the observed data.

A proper scoring rule, S, takes as input a forecasted probability distribution, P, and one observation y(outcome), and assigns a score S(P,y) to the forecast such that the true distribution of the outcomes gets the best score in expectation.

The most commonly used proper scoring rule is the logarithmic score L, which, when minimized we get the MLE

$$\mathcal{L}(\theta, y) = -log\mathcal{P}_\theta(y)$$

which is nothing but the log likelihood that we have seen in so many places. And the scoring rule is parametrized by θ because that is what we are predicting as part of the machine learning model.

Another example is CRPS(Continuous Ranked Probability Score). While the logarithmic score or the log likelihood generalizes Mean Squared Error to a probabilistic space, CRPS does the same to Mean Absolute Error.

$$\mathcal{C}(\theta, y) = \int_{-\infty}^{y} F_\theta(z)^2 dz + \int_{y}^{\infty} (1 - F_\theta(z))^2 dz$$

3 Optimization

When working on a regular regression problem, we often minimize mean squared error. How do we pick a loss function if what we are trying to predict is a probability distribution? In this case we use a scoring rule. A proper scoring rule S takes as input a forecasted probability distribution P and one observation y (outcome), and assigns a score S(P, y) to the forecast such that the true distribution of the outcomes gets the best score in expectation. For many problems, we can use negative log-likelihood as a scoring function.

In NLL, what we are trying to get is how likely is a target value y given the distribution parameterized with θ. The scoring rule for the best choice of the parameters θ is the smallest of the scoring rules. The difference between the scoring rules is called a divergence and is a measure of the "distance" between probability distributions. If Q is the true distribution and P is a predicted distribution the divergence is:

$$D_{\mathcal{S}}(Q\|P) = \mathbb{E}_{y\sim Q}[\mathcal{S}(P, y)] - \mathbb{E}_{y\sim Q}[\mathcal{S}(Q, y)]$$

If we use MLE, then the divergence becomes Kullback–Leibler divergence, widely used to measure the difference between probability distributions.

3.1 Natural Gradient

"NG" in NGBoost stands for "Natural Gradient". (Or could it be because it is the last name of one of the co-author of the paper Andrew Ng?) Why did it have to be introduced?

Once you start using gradient-based methods in order to learn parameters θ, you quickly realize that the distance between two distribution is not defined by the difference of their parameters, and so direction of the gradient by the parameters does not necessarily indicate the direction of the best improvement of the model.

The problem is that "distance" between two parameter values does not correspond to an appropriate "distance" between the distributions that those parameters identify.

The Natural Gradient corrects this issue by using the divergence instead of parameter difference. The generalized natural gradient is the direction of steepest ascent in Riemannian space, which is invariant to parametrization, and is defined as:

$$\tilde{\nabla}\mathcal{S}(\theta, y) \propto \lim_{\epsilon\to 0} \underset{d:D_{\mathcal{S}}(P_\theta||P_{\theta+d})=\epsilon}{\arg\max} \mathcal{S}(\theta + d, y).$$

So, instead of drawing a unit sphere in the parameter space around the point and finding the direction of maximum improvement, we draw a surface on which the divergence takes the same value ϵ. The natural gradient is connected with the regular gradient through the Riemann metric of the statistical manifold:

$$\tilde{\nabla}\mathcal{S}(\theta, y) \quad \propto \quad \mathcal{I}_{\mathcal{S}}(\theta)^{-1}\nabla\mathcal{S}(\theta, y)$$

If we use MLE, the matrix I is the Fisher Information matrix.

4 NGBoost

4.1 Putting it all together

Now that we have seen the major components, let's take a look at how all of this works together. NGBoost is a supervised learning method for probabilistic prediction that uses boosting to estimate the parameters of a conditional probability distribution P(y|x). As we saw earlier, we need to choose three modular components upfront:

- Base learner (f)
- Parametric Probability Distribution (P parameterized by θ)
- Proper scoring rule (S)

A prediction y|x on a new input x is made in the form of a conditional distribution P, whose parameters θ are obtained by an additive combination of M base learners outputs and in initial $\theta 0$. Let's denote the combined function learned by the M base learners for all parameters by f. And there will be a separate set of base learners for each parameter in the chosen probability distribution. For eg. in the normal distribution, there will be f^m for μ and f^m for $\log\sigma$. The predicted outputs are also scaled with a stage specific scaling factor ρ^m , and a common learning rate η:

$$y|x \sim \mathcal{P}\theta(x), \theta = \theta_0 + \eta \sum_{m=1}^{M} \rho^{(m)} \cdot f^{(m)}(x)$$

One thing to note here is that even if you have n parameters for your probability distribution, ρ is still a single scalar.

4.1.1 Algorithm

Let's look at the algorithm as explained in the paper

Algorithm 1 NGBoost for probabilistic prediction

Data: Dataset $\mathcal{D} = \{x_i, y_i\}_{i=1}^n$.
Input: Boosting iterations M, Learning rate η, Probability distribution with parameter θ, Proper scoring rule $\mathcal{S}$, Base learner f.
Output: Scalings and base learners $\{\rho^{(m)}, f^{(m)}\}_{m=1}^M$.

$\theta^{(0)} \leftarrow \arg\min_\theta \sum_{i=1}^n \mathcal{S}(\theta, y_i)$ {initialize to marginal}
for $m \leftarrow 1, \ldots, M$ **do**
  for $i \leftarrow 1, \ldots, n$ **do**
    $g_i^{(m)} \leftarrow \mathcal{I}_\mathcal{S}\left(\theta_i^{(m-1)}\right)^{-1} \nabla_\theta \mathcal{S}\left(\theta_i^{(m-1)}, y_i\right)$
  end
  $f^{(m)} \leftarrow \text{fit}\left(\left\{x_i, g_i^{(m)}\right\}_{i=1}^n\right)$
  $\rho^{(m)} \leftarrow \arg\min_\rho \sum_{i=1}^n \mathcal{S}\left(\theta_i^{(m-1)} - \rho \cdot f^{(m)}(x_i), y_i\right)$
  for $i \leftarrow 1, \ldots, n$ **do**
    $\theta_i^{(m)} \leftarrow \theta_i^{(m-1)} - \eta\left(\rho^{(m)} \cdot f^{(m)}(x_i)\right)$
  end
end

Let us consider a Dataset

$$\mathcal{D} = x_i, y_{i i=1}^{n}$$

Boosting Iterations M, Learning rate η, Probability Distribution with parameters θ, Proper scoring rule S, and Base learner f

Initialize $\theta 0$ to the marginal. This is just estimating the parameters of the distribution without conditioning on any covariates; similar to initializing to mean. Mathematically, we solve this equation:

$$argmin_\theta \sum_{i=1}^{n} S(\theta, y_i)$$

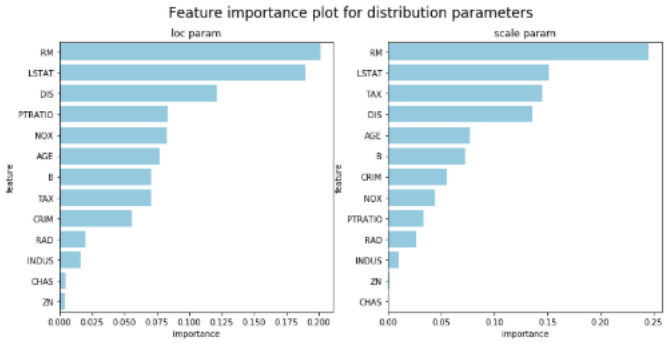
- For each iteration in M:
- Calculate the Natural gradients g ^{m} of the Scoring rule S with respect to the parameters at previous stage $^{m-1}$ for all n examples in dataset.
- A set of base learners for that iteration f^m are fit to predict the corresponding components of the natural gradients, g^m. This output can be thought of as the projection of the natural gradient on to the range of the base learner class, because we are training the base learners to predict the natural gradient at current stage.
- This projected gradient is then scaled by a scaling factor ρ ^{m}. This is because Natural Gradients rely on local approximations(as we saw in the earlier post) and these local approximations won't hold good far away from the current parameter position.

  In practice, we use a line search to get the best scaling factor which minimizes the overall scoring rule. In the implementation, they have found out that reducing the scaling factor by half in the line search works well.
- Once the scaling parameter is estimated, we update the parameters by adding the negative scaled projected gradient to the outputs of previous stage, after further scaling by a learning rate.

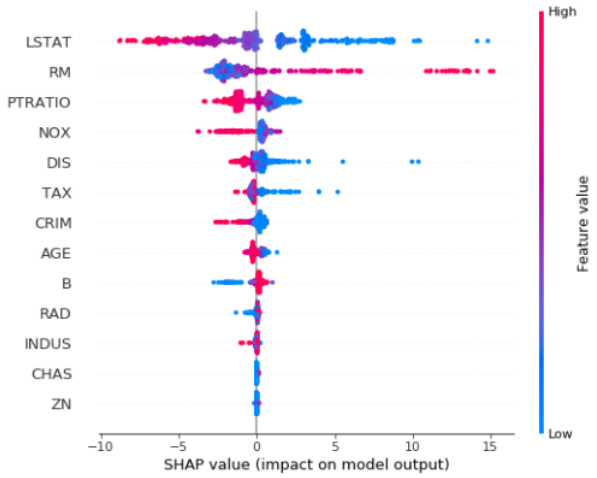
$$\theta_i^{(m)} \leftarrow \theta_i^{(m-1)} - \eta(\rho^{(m)} \cdot f^{(m)}(x_i))$$

5 Interpretation of NGBoost

Although there needs to be a considerable amount of caution before using the importances from machine learning models, NGBoost also offers feature importances. It has separate set of importances for each parameter it estimates.



But the best part is not just this, but that SHAP, is also readily available for the model. You just need to use TreeExplainer to get the values.



5.1 Experiments

The paper also looks at how the algorithm performs when compared to other popular algorithms. There were two separate type of evaluation — Probabilistic, and Point Estimation

5.2 Probabilistic Regression

On a variety of datasets from the UCI Machine Learning Repository, NGBoost was compared with other major probabilistic regression algorithms, like Monte-Carlo Dropout, Deep Ensembles, Concrete Dropout, Gaussian Process, Generalized Additive Model for Location, Scale and Shape(GAMLSS), Distributional Forests.

For all experiments, NGBoost was configured with the Normal distribution, decision tree base learner with a maximum depth of three levels, and log scoring rule. In general we recommend small learning rates, subject to computational feasibility.

Table 1. Comparison of probabilistic regression performance on regression benchmark UCI datasets as measured by NLL. Results for MC dropout, Deep Ensembles, and Concrete Dropout are reported from Gal and Ghahramani (2016); Lakshminarayanan et al. (2017); Gal et al. (2017) respectively. NGBoost offers competitive performance in terms of NLL, especially on smaller datasets. The best method for each dataset is bolded, as are those with standard errors that overlap with the best method.

Dataset	N	NGBoost	MC dropout	Deep Ensembles	Concrete Dropout	Gaussian Process	GAMLSS	DistForest
Boston	506	**2.43 ± 0.15**	**2.46 ± 0.25**	**2.41 ± 0.25**	2.72 ± 0.01	**2.37 ± 0.24**	2.73 ± 0.56	2.67 ± 0.08
Concrete	1030	**3.04 ± 0.17**	**3.04 ± 0.09**	**3.06 ± 0.18**	3.51 ± 0.00	**3.03 ± 0.11**	3.24 ± 0.08	3.38 ± 0.05
Energy	768	**0.60 ± 0.45**	1.99 ± 0.09	1.38 ± 0.22	2.30 ± 0.00	**0.66 ± 0.17**	1.24 ± 0.86	1.53 ± 0.14
Kin8nm	8192	-0.49 ± 0.02	-0.95 ± 0.03	**-1.20 ± 0.02**	-0.65 ± 0.00	-1.11 ± 0.03	-0.26 ± 0.02	-0.40 ± 0.01
Naval	11934	-5.34± 0.04	-3.80 ± 0.05	-5.63 ± 0.05	**-5.87 ± 0.05**	-4.98 ± 0.02	-5.56 ± 0.07	-4.84 ± 0.01
Power	9568	2.79 ± 0.11	2.80 ± 0.05	2.79 ± 0.04	2.75 ± 0.01	2.81 ± 0.05	2.86 ± 0.04	**2.68 ± 0.05**
Protein	45730	2.81 ± 0.03	2.89 ± 0.01	2.83 ± 0.02	2.81 ± 0.00	2.89 ± 0.02	3.00 ± 0.01	**2.59 ± 0.04**
Wine	1588	**0.91 ± 0.06**	**0.93 ± 0.06**	**0.94 ± 0.12**	1.70 ± 0.00	**0.95 ± 0.06**	**0.97 ± 0.09**	1.05 ± 0.15
Yacht	308	**0.20 ± 0.26**	1.55 ± 0.12	1.18 ± 0.21	1.75 ± 0.00	**0.10 ± 0.26**	0.80 ± 0.56	2.94 ± 0.09
Year MSD	515345	3.43 ± NA	3.59 ± NA	**3.35 ± NA**	NA± NA	NA ± NA	NA ± NA	NA ± NA

5.3 Point Estimate

They also evaluated the algorithm on the point estimate use case against other regression algorithms like Elastic Net, Random Forest(Sci-kit Learn), Gradient Boosting(Sci-kit Learn). Although NGBoost is not specifically designed for point estimation, it is easy to extract point estimates of expectations from the estimated distributions. They use this approach in a third evaluation to compare the same NGBoost models as above to the Scikit-Learn implementations of random forests, standard gradient boosting, and elastic net regression.

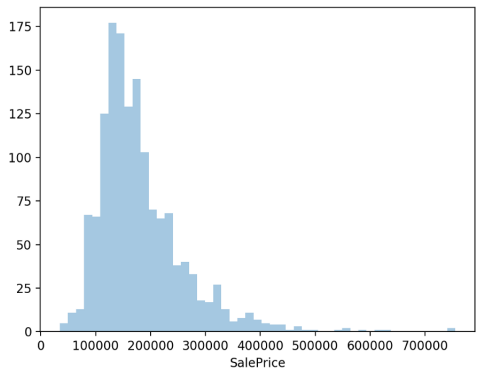
Table 3. Comparison of point-estimation performance on regression benchmark UCI datasets as measured by RMSE. Although not optimized for point estimation, NGBoost still offers competitive performance. Bolding is as in Table 1.

Dataset	N	NGBoost	Elastic Net	Random Forest	Gradient Boosting	GAMLSS	Distributional Forest
Boston	506	**2.94 ± 0.53**	4.08 ± 0.16	2.97 ± 0.30	**2.46 ± 0.32**	4.32 ± 1.40	3.99 ± 1.13
Concrete	1030	**5.06 ± 0.61**	12.1 ± 0.05	5.29 ± 0.16	**4.46 ± 0.29**	6.72 ± 0.59	6.61 ± 0.83
Energy	768	0.46 ± 0.06	2.75 ± 0.03	0.52 ± 0.09	**0.39 ± 0.02**	1.43 ± 0.32	1.11 ± 0.27
Kin8nm	8192	0.16 ± 0.00	0.20 ± 0.00	0.15 ± 0.00	**0.14 ± 0.00**	0.20 ± 0.01	0.16 ± 0.00
Naval	11934	**0.00 ± 0.00**	**0.00 ± 0.00**	**0.00 ± 0.00**	**0.00 ± 0.00**	**0.00 ± 0.00**	**0.00 ± 0.00**
Power	9568	3.79 ± 0.18	4.42 ± 0.00	3.26 ± 0.03	**3.01 ± 0.10**	4.25 ± 0.19	3.64 ± 0.24
Protein	45730	4.33 ± 0.03	5.20 ± 0.00	**3.60 ± 0.00**	3.95 ± 0.00	5.04 ± 0.04	3.89 ± 0.04
Wine	1588	0.63 ± 0.04	0.58 ± 0.00	**0.50 ± 0.01**	0.53 ± 0.02	0.64 ± 0.04	0.67 ± 0.05
Yacht	308	**0.50 ± 0.20**	7.65 ± 0.21	0.61 ± 0.08	**0.42 ± 0.09**	8.29 ± 2.56	4.19± 0.92
Year MSD	515345	8.94 ± NA	9.49 ± NA	9.05 ± NA	**8.73 ± NA**	NA ± NA	NA ± NA

5.4 Empirical Validation — NGBoost Comparison to LightGBM and XGBoost

Let's implement NGBoost and see how is the performance of it. The original paper also did some experiments on various datasets. They compared MC dropout, Deep Ensembles and NGBoost in regression problems and NGBoost shows its quite competitive performance. In this blog post, I would like to show the model performance on the famous house price prediction dataset on Kaggle. This dataset consists of 81 features, 1460 rows and the target feature is the sale price. Let's see NGBoost can handle these conditions.

The Distribution of target feature:



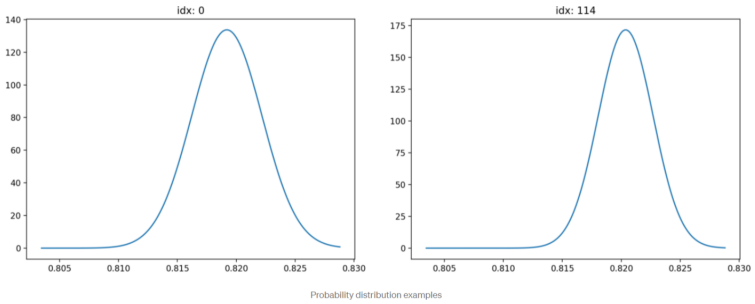
We take the best perfomance result of lightGBM and XGboost from kaggle website and compare it with our best performing NGBoost algorithm.

Here is the summary of prediction losses (RMSE):

RMSE		
NGBoost	LightGBM	XGBoost
0.0034	0.0049	0.0036

It seems like NGBoost outperformed other famous boosting algorithms. To be fair, I feel like if I tune the parameters of BGBoost, it will be even better.

NGBoost's one of the biggest difference from other boosting algorithms is can return probabilistic distribution of each prediction. This can be visualised by using pred_dist function. This function enables to show the results of probabilistic predictions.



Above plots are the probability distributions of each prediction. X-axis shows the log value of Sale Price (target feature). We can observe that the probability distribution is wider for index 0 than index 114.

6 Conclusion

Predicting a conditional distribution of a continuous variable is hard, it is much easier to predict a point estimate. It is different in classification problems, where more algorithms are able to predict the class probability. The NGBoost algorithm allows to easily get prediction of the parameters of conditional distribution of the target variable given the predictors. There are few assumptions on the type of distribution that can be handled by NGBoost. Indeed, if Normal distribution is not fit for a given problem, another distribution can be chosen (with potentially more parameters) to fit the model better.

But one drawback here is with the performance of the algorithm. The time complexity linearly increases with each additional parameter we have to estimate. This can be probably improved by using subsampling method. And all the efficiency hacks/changes which has made its way into popular Gradient Boosting packages like LightGBM, or XGBoost are not present in the current implementation. Maybe it would be ported over soon enough because I see the repo is under active development and see this as one of the action items they are targeting. For example there's no early stopping option, no option of showing the intermediate results, the flexibility of choosing the base learner (so far we can only choose between decision tree and Ridge regression), setting a random state seed, and so on. I believe these points will be implemented very soon. But until that happens, this is quite slow, especially with large data. One way out is to use minibatch parameter to make the calculation of natural gradients faster.

References

[1] T. Duan, et al., NGBoost: Natural Gradient Boosting for Probabilistic Prediction (2019), ArXiv 1910.03225