Abstract

Gradient-boosted regression trees (GBRTs) are hugely popular for solving tabular regression problems, but provide no estimate of uncertainty. We propose Instance-Based Uncertainty estimation for Gradient-boosted regression trees (IBUG), a simple method for extending any GBRT point predictor to produce probabilistic predictions. IBUG computes a non-parametric distribution around a prediction using the \( k \)-nearest training instances, where distance is measured with a tree-ensemble kernel. The runtime of IBUG depends on the number of training examples at each leaf in the ensemble, and can be improved by sampling trees or training instances. Empirically, we find that IBUG achieves similar or better performance than the previous state-of-the-art across 22 benchmark regression datasets. We also find that IBUG can achieve improved probabilistic performance by using different base GBRT models, and can more flexibly model the posterior distribution of a prediction than competing methods. We also find that previous methods suffer from poor probabilistic calibration on some datasets, which can be mitigated using a scalar factor tuned on the validation data. Source code is available at https://github.com/jjbrophy47/ibug.

1 Introduction

Despite the impressive success of deep learning models on unstructured data (e.g., images, audio, text), gradient-boosted trees \([26]\) remain the preferred choice for tabular or structured data \([52]\). In fact, Kaggle CEO Anthony Goldbloom recently described gradient-boosted trees as the most “glaring difference” between what is used on Kaggle and what is “fashionable in academia” \([34]\).

Our focus is on tabular data for regression tasks, which vary widely from financial \([1]\) and retail-product forecasting \([43]\) to weather \([29, 28]\) and clinic-mortality prediction \([6]\). Gradient-boosted regression trees (GBRTs) are known to make accurate point predictions \([42]\) but provide no estimate of the prediction uncertainty, which is desirable for both forecasting practitioners \([9, 68]\) and the explainable AI (XAI) community \([25, 69, 2]\) in general. Recently, Duan et al. \([22]\), Sprangers et al. \([64]\), and Malinin et al. \([44]\) introduced NGBoost, PGBM, and CBU (CatBoost \([52]\) with uncertainty), respectively, new gradient boosting algorithms that provide state-of-the-art probabilistic predictions. However, NGBoost tends to underperform as a point-predictor, and PGBM and CBU are limited in the types of distributions they can use to model the output.

We introduce a simple yet effective method for enabling any GBRT point-prediction model to produce probabilistic predictions. Our proposed approach, Instance-Based Uncertainty estimation for Gradient-boosted regression trees \((IBUG)\), has two key components: 1) We leverage the fact that GBRTs accurately model the conditional mean and use this point prediction as the mean in a probabilistic forecast; and 2) We identify the \( k \) training examples with the greatest affinity to the

*https://www.jonathanbrophy.com

target instance and use these examples to estimate the uncertainty of the target prediction. We define the affinity between two instances as the number of times both instances appear in the same leaf throughout the ensemble. Thus, our method acts as a wrapper around any given GBRT model.

In experiments on 21 regression benchmark datasets and one synthetic dataset, we demonstrate the effectiveness of IBUG to deliver on par or improved probabilistic performance as compared to existing state-of-the-art methods while maintaining state-of-the-art point-prediction performance. We also show that probabilistic predictions can be improved by applying IBUG to different GBRT models, something that NGBoost and PGBM cannot do. Additionally, IBUG can use the training instances closest to the target example to directly model the output distribution using any parametric or non-parametric distribution, something NGBoost, PGBM, and CBU cannot do. Finally, we show that sampling trees dramatically improves runtime efficiency for computing training-example affinities without having a significant detrimental impact on the resulting probabilistic predictions, allowing IBUG to scale to larger datasets.

2 Notation & Background

We assume an instance space $\mathcal{X} \subseteq \mathbb{R}^p$ and target space $\mathcal{Y} \subset \mathbb{R}$. Let $\mathcal{D} := \{(x_i, y_i)\}_{i=1}^n$ be a training dataset in which each instance $x_i \in \mathcal{X}$ is a $p$-dimensional vector $(x_i^j)_{j=1}^p$ and $y_i \in \mathcal{Y}$.

2.1 Gradient-Boosted Regression Trees

Gradient-boosting \cite{friedman2001greedy} is a powerful machine-learning algorithm that iteratively adds weak learners to construct a model $f : \mathcal{X} \to \mathbb{R}$ that minimizes some empirical risk $\mathcal{L} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. The model is defined by a recursive relationship: $f_0(x) = \gamma, \ldots, f_t(x) = f_{t-1}(x) + \eta m_t(x)$ in which $f_0$ is the base learner, $\gamma$ is an initial estimate, $f_i$ is the model at iteration $t$, $m_t$ is the weak learner added during iteration $t$ to improve the model, and $\eta$ is the learning rate.

Gradient-boosted regression trees (GBRTs) typically choose $\ell$ to be the mean squared error (MSE), $\gamma$ as $\frac{1}{n} \sum_{i=1}^{n} y_i$ (mean output of the training instances), and regression trees as weak learners. Each weak learner is typically chosen to approximate the negative gradient \cite{hastie2001elements}: $m_t = \arg \min_{m} \frac{1}{n} \sum_{i=1}^{n} (-g_i^t - m(x_i))^2$ in which $g_i^t = \frac{\partial \ell(y_i, \hat{y}_i)}{\partial \hat{y}_i}$ is the functional gradient of the $i$th training instance at iteration $t$, with respect to $\hat{y}_i = f_{t-1}(x_i)$.

The weak learner at iteration $t$ recursively partitions the instance space into $M_t$ disjoint regions $\{r_t^j\}_{j=1}^{M_t}$. Each region is called a leaf, and the parameter value for leaf $j$ at tree $t$ is typically determined (given a fixed structure) using a one-step Newton-estimation method \cite{friedman2001greedy}: $\theta_t^j = -\sum_{i \in r_t^j} g_i^t \big( \sum_{i \in r_t^j} h_t^i + \lambda \big)$ in which $I_t^j = \{(x_i, y_i) \mid x_i \in r_t^j\}_{i=1}^n$ is the instance set of leaf $j$ for tree $t$, $h_t^j$ is the second derivative of the $i$th training instance w.r.t. $\hat{y}_i$, and $\lambda$ is a regularization constant. Thus, the output of $m_t$ can be written as follows: $m_t(x) = \sum_{j=1}^{M_t} \theta_t^j I[x \in r_t^j]$ in which $I$ is the indicator function. The final GBRT model generates a prediction for a target example $x_{te}$ by summing the values of the leaves $x_{te}$ traverses to across all $T$ iterations: $\hat{y}_{te} = \sum_{t=1}^{T} m_t(x_{te})$.

2.2 Probabilistic Regression

Our focus is on probabilistic $y$ given some input vector $x \in \mathcal{X}$.Unfortunately, traditional GBRT models only output scalar values. Under a squared-error loss function, these scalar values can be interpreted as the conditional mean in a Gaussian distribution with some (unknown) constant variance. However, homoscedasticity is a strong assumption and unknown constant variance has little value in a probabilistic prediction; thus, in order to allow heteroscedasticity, the predicted distribution needs at least two parameters to convey both the magnitude and uncertainty of the prediction \cite{duan2019natural}.

Natural Gradient Boosting (NGBoost) is a recent method by Duan et al. \cite{duan2019natural} that tackles the aforementioned problems by estimating the parameters of a desired distribution using a multi-parameter boosting approach that trains a separate ensemble for each parameter of the distribution. NGBoost employs the natural gradient to be invariant to parameterization, but requires the inversion...
of many small matrices (each the size of the number of parameters) to do so. Empirically, NGBoost generates state-of-the-art probabilistic predictions, but tends to underperform as a point predictor.

More recently, Sprangers et al. [64] introduced Probabilistic Gradient Boosting Machines (PGBM), a single model that optimizes for point performance, but can also generate accurate probabilistic predictions. PGBM treats leaf values as stochastic random variables, using sample statistics to model the mean and variance of each leaf value. PGBM estimates the output mean and variance of a target example using the estimated parameters of each leaf it is assigned to. The predicted mean and variance are then used as parameters in a specified distribution to generate a probabilistic prediction. PGBM has been shown to produce state-of-the-art probabilistic predictions; however, computing the necessary leaf statistics during training can be computationally expensive, especially as the number of leaves in the ensemble increases (see §5.6). Also, since only the mean and variance are predicted for a given target example, PGBM is limited to distributions using only location and scale to model the output.

Finally, Malinin et al. [44] introduce CatBoost with uncertainty (CBU), a method that estimates uncertainty using ensembles of GBRT models. Similar to NGBoost, multiple ensembles are learned to output the mean and variance. However, CBU also constructs a virtual ensemble—a set of overlapping partitions of the learned GBRT trees—to estimate the uncertainty of a prediction by taking the mean of the variances output from the virtual ensemble. Their approach uses a recently proposed stochastic gradient Langevin boosting algorithm [72] to sample from the true posterior via the virtual ensemble (in the limit); however, their formulation of uncertainty is limited only to the first and second moments, similar to PGBM.

To address the shortcomings of existing approaches, we introduce a simple method that performs well on both point and probabilistic performance, and can more flexibly model the output than previous approaches. Our method can also be applied to any GBRT model, adding additional flexibility.

3 Instance-Based Uncertainty

Instance-based methods such as $k$-nearest neighbors have been around for decades and have been useful for many different machine learning tasks [50]. However, defining neighbors based on a fixed metric like Euclidean distance may lead to suboptimal performance, especially as the dimensionality of the dataset increases. More recently, it has been shown that random forests can be used as an adaptive nearest neighbors method [18, 40] which identifies the most similar examples to a given instance using the learned model structure. This supervised tree kernel can more effectively measure the similarity between examples, and has been used for clustering [46] and local linear modeling [8] as well as instance- and feature-based attribution explanations [51], for example.

In this work, we apply the idea of a supervised tree kernel to help model the uncertainty of a given GBRT prediction. Our approach, Instance-Based Uncertainty estimation for Gradient-boosted regression trees (IBUG), identifies the neighborhood of similar training examples to a target example using the structure of the GBRT, and then uses those instances to generate a probabilistic prediction. IBUG works for any GBRT, and can more flexibly model the output than competing methods.

3.1 Identification of High-Affinity Neighbors

At its core, IBUG uses the $k$ training examples with the largest affinity to the target example to model the conditional output distribution. Given a GBRT model $f$, we define the affinity between two examples simply as the number of times each instance appears in the same leaf across all trees in $f$. Thus, the affinity of the $i$th training example $x_i$ to a target example $x_{te}$ can be written as:

$$A(x_i, x_{te}) = \sum_{t=1}^{T} \mathbb{I}[R_t(x_i) = R_t(x_{te})],$$

where $R_t(x_i)$ is the leaf $x_i$ is assigned to for tree $t$. Alg. 1 summarizes the procedure for computing affinity scores for all training examples. This metric clusters similar examples together based on the learned model representation (i.e., the tree structures). Intuitively, if two examples appear in the same leaf in every tree throughout the ensemble, then both examples are predicted in an identical manner. One may also view Eq. (1) as an indication of which training examples most often affect the leaf values $x_{te}$ is assigned to and thus implicitly which examples are likely to have a big effect on the
prediction $\hat{y}_{te}$. This similarity metric is similar to the random forest kernel [18], however, unlike random forests, GBRTs are typically constructed to a shallower depth, resulting in more training examples assigned to the same leaf (see §C.5 for additional details about leaf density in GBRTs).

Algorithm 1 IBUG affinity computation.

\begin{algorithm}
\caption{IBUG affinity computation.}
\label{alg:IBUG_affinity}
\begin{algorithmic}[1]
\Procedure{ComputeAffinities}{x, f}
\State $A \leftarrow 0$ \Comment{Init. train affinities}
\For{$t = 1 \ldots T$} \Comment{Visit each tree}
\State Get instance set $I_t$ for leaf $l = R_t(x)$
\For{$i \in I_t^l$} \Comment{Increment affinities}
\State $A_i \leftarrow A_i + 1$
\EndFor
\EndFor
\State \Return $A$
\EndProcedure
\end{algorithmic}
\end{algorithm}

Algorithm 2 IBUG probabilistic prediction.

\begin{algorithm}
\caption{IBUG probabilistic prediction.}
\label{alg:IBUG_prob_pred}
\begin{algorithmic}[1]
\Procedure{ProbPredict}{x, $A^{(k)}$, $\rho, \gamma, \delta$}
\State $\mu_0 \leftarrow f(x)$ \Comment{GBRT scalar output}
\State $\sigma_0^2 \leftarrow \max(\sigma^2(A^{(k)}), \rho)$ \Comment{Ensure $\sigma^2 > 0$}
\State $\sigma_0^2 \leftarrow \gamma \sigma_0^2 + \delta$ \Comment{Var. calibration, Eq. (2)}
\State \Return $D(A^{(k)}|\mu_0, \sigma_0^2)$ \Comment{Eq. (3)}
\EndProcedure
\end{algorithmic}
\end{algorithm}

3.2 Modeling the Output Distribution

IBUG has a multitude of choices when modeling the conditional output distribution. The simplest and most common approach is to model the output assuming a Gaussian distribution [22] [64]. We use the scalar output of $f: \mu_0 = f(x_{te})$ to model the conditional mean since GBRTs already produce accurate point predictions. Then, we use the $k$ training instances with the largest affinity to $x_{te}$—we denote this set $A^{(k)}$—to compute the variance $\sigma_0^2$.

**Calibrating prediction variance.** The $k$-nearest neighbors generally do a good job of determining the relative uncertainty of different predictions, but on some datasets, the resulting variance is systematically too large or too small. To correct for this, we apply an additional affine transformation before making the prediction:

$$\sigma_{y_{te}}^2 \leftarrow \gamma \sigma_{y_{te}}^2 + \delta,$$  \hspace{1cm} (2)

where $\gamma$ and $\delta$ are tuned on validation data after $k$ has been selected. Instead of exhaustively searching over all values of $\gamma$ or $\delta$, we use either the multiplicative factor (tuning $\gamma$ with $\delta = 0$) or the additive factor (tuning $\delta$ with $\gamma = 1$), and choose between them using their performance on validation data.

We find this simple calibration step consistently improves probabilistic performance for not only IBUG, but competing methods as well, and at a relatively small cost compared to training the model.

**Flexible posterior modeling.** In general, we can generate a probabilistic prediction using $\mu_0$ and $\sigma_{y_{te}}^2$ for any distribution that uses location and scale (note PGBM and CBU can only model these types of distributions). However, IBUG can additionally use $A^{(k)}$ to directly fit any continuous distribution $D$, including those with high-order moments:

$$\tilde{D}_{te} = D(A^{(k)}|\mu_{y_{te}}, \sigma_{y_{te}}^2).$$  \hspace{1cm} (3)

Eq. (3) is defined such that $D$ can be fit directly with $A^{(k)}$ using MLE (maximum likelihood estimation) [47], or may be fit using $\mu_{y_{te}}$ or $\sigma_{y_{te}}^2$ as fixed parameter values with $A^{(k)}$ fitting any other parameters of the distribution. Overall, directly fitting all or some additional parameters in $D$—for example, the shape parameter in a Weibull distribution—is a benefit over PGBM and CBU, which can only optimize for a *global* shape value using a gridsearch-like approach with extra validation data.

Note that NGBoost can model any parameterized distribution, but must specify this choice before training; in contrast, IBUG can optimize this choice after training. Additionally, IBUG may choose $D$ to be a *non-parametric density estimator* such as KDE (kernel density estimation) [62], which PGBM, CBU, and NGBoost cannot do.

3.3 Summary

In summary, Alg. 2 provides pseudocode for generating a probabilistic prediction with IBUG. Note Algs. 1 and 2 work for any GBRT model, allowing practitioners to employ IBUG to adapt multiple different point predictors into probabilistic estimators and select the model with the best performance.
Empirically, we show using different base models for IBUG can result in improved probabilistic performance than using just one (§5.3).

IBUG is a nearest neighbors approach and thus seems well-suited to estimating aleatoric uncertainty—remaining uncertainty due to irreducible error or the inherent stochasticity in the system [33]—since it can quantify the range of outcomes to be expected given the observed features. However, we use predictions on held-out data to tune the number of nearest neighbors and the variance calibration hyperparameters; thus, we effectively optimize prediction uncertainty encompassing both aleatoric uncertainty and epistemic uncertainty—error due to the imperfections of the model and the training data [20, 44]. The evaluation measures in our experiments thus also focus on predictive uncertainty.

4 Computational Efficiency

Training efficiency. Since IBUG works with standard GBRT models, it inherits the training efficiency of modern GBRT implementations such as XGBoost [12], LightGBM [36], and CatBoost [52].

Prediction efficiency. If there are $T$ trees in the ensemble and each leaf has at most $n_l$ training instances assigned to it, then IBUG’s prediction time is $O(Tn_l)$, since it considers each instance in each leaf. Note training instances that do not appear in a leaf with the target instance do not increase prediction time; what matters most is thus the number of instances at each leaf. We find LightGBM often induces regression trees with large leaves—in some cases, over half the dataset is assigned to a single leaf (see §C.5 for details). Thus, prediction time still grows with the size of the dataset, as is typical for instance-based methods. This higher prediction time is the price IBUG pays for greater flexibility.

Prediction efficiency can be increased at training time by using deeper GBRTs with fewer instances in each leaf, after training by subsampling the instances considered for predictions, or at prediction time by sampling the trees used to compute affinities. We explore this last option in the next subsection.

4.1 Sampling Trees

The most expensive operation when generating a probabilistic prediction with IBUG is computing the affinity vector (Eq. 1). In order to increase prediction efficiency, we can instead work with a subset of the trees $\tau < T$ in the ensemble. We can build this subset by sampling trees uniformly at random, taking the first trees learned (representing the largest gradient steps), or the last trees learned (representing the fine-tuning steps).

By sampling trees, the runtime complexity reduces to $O(\tau n_l)$, which provides significant speedups when $\tau \ll T$. In our empirical evaluation, we find that taking a subset of the first trees learned generally works best, significantly increasing prediction efficiency while maintaining accurate probabilistic predictions (§5.6).

4.2 Accelerated $k$ Tuning

Choosing an appropriate value of $k$ is critical for generating accurate probabilistic predictions in IBUG. Thus, we aim to tune $k$ using a held-out validation dataset $D_{val} \subset D$ and an appropriate probabilistic scoring metric such as negative log likelihood (NLL). Unfortunately, typical tuning procedures would result in the same affinity vectors being computed—an expensive operation—for each candidate value of $k$. To mitigate this issue, we perform a custom tuning procedure that reuses computed affinity vectors for all values of $k$. More specifically, IBUG computes an affinity vector $A$ for a given validation example $x_{val}$, and then sorts $A$ in descending order (i.e., largest affinity first). Then, IBUG takes the top $k$ training instances, and generates and scores the resulting probabilistic prediction. For each subsequent value of $k$, the same sorted affinity list can be used, avoiding duplicate computation. We summarize this procedure in Alg. [3].

Once $k$ is chosen, we may encounter a new unseen target instance in which the variance of the $k$-highest affinity training examples for that target example is zero or extremely small. In this case, we set the predicted target variance to $\rho$, which is set during tuning to the minimum (nonzero) variance computed over all predictions in the validation set for the chosen $k$. In practice, we find instances of abnormally low variance to be rare with appropriately chosen values of $k$. 

5
Algorithm 3 IBUG accelerated tuning of $k$.

**Input:** Validation dataset $D_{val} \subseteq D$, GBRT model $f$, list of candidates $K$, target distribution $D$, probabilistic scoring metric $V$, minimum variance $\rho = 1e^{-15}$.

```
1: procedure FASTTUNE($D_{val}, f, K, D, V, \rho$)
2:   for $(x_j, y_j) \in D_{val}$ do
3:     $A \leftarrow$ COMPUTEAFFINITIES($x_j, f$)          ▷ Algorithm 1
4:     $A \leftarrow$ Argsort $A$ in descending order
5:     for $k \in K$ do                                   ▷ Use same ordering for each $k$
6:      $A(k) \leftarrow$ Take first $k$ training instances($A, k$)
7:      $\hat{D}_y \leftarrow$ PROBPREDICT($x_j, f, A(k), \rho, 1, 0, D$) ▷ Algorithm 2
8:      $S_k \leftarrow V(y_j, \hat{D}_y)$                  ▷ Save validation score
9:     $k \leftarrow$ Select best $k$ from $S$
10:    $\rho \leftarrow$ Select minimum $\sigma^2$ from $\hat{D}_y$
11: return $k, \rho$
```

5 Experiments

In this section, we demonstrate IBUG’s ability to produce competitive probabilistic and point predictions as compared to current state-of-the-art methods on a large set of regression datasets (§5.1, §5.2). Then, we show that IBUG can use different base models to improve probabilistic performance (§5.3), flexibly model the posterior distribution (§5.4), and use approximations to speed up probabilistic predictions while maintaining competitive performance (§5.6).

Implementation and Reproducibility. We implement IBUG in Python, using Cython—a Python package allowing the development of C extensions—to store a unified representation of the model structure. IBUG supports all modern gradient boosting frameworks including XGBoost [12], LightGBM [36], and CatBoost [52]. Experiments are run on publicly available datasets using an Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.6GHz with 60GB of RAM @ 2.4GHz. Links to all data sources as well as the code for IBUG and all experiments is available at [https://github.com/jjbrophy47/ibug](https://github.com/jjbrophy47/ibug).

5.1 Methodology

We now compare IBUG’s probabilistic and point predictions to NGBoost [22], PGBM [64], and CBU [44] on 21 benchmark regression datasets and one synthetic dataset. Additional dataset details are in §B.1.

Metrics. We compute the average continuous ranked probability score (CRPS ↓) and negative log likelihood (NLL ↓) [29, 76] over the test set to evaluate probabilistic performance. To evaluate point performance, we use root mean squared error (RMSE ↓). For all metrics, lower is better. See §B for detailed descriptions.

Protocol. We follow a similar protocol to Sprangers et al. [64] and Duan et al. [22]. We use 10-fold cross-validation to create 10 90/10 train/test folds for each dataset. For each fold, the 90% training set is randomly split into an 80/20 train/validation set to tune any hyperparameters. Once the hyperparameters are tuned, the model is retrained using the entire 90% training set. For probabilistic predictions, a normal distribution is used to model the output.

Significance Testing. We report counts of the number of datasets in which a given method performed better (“Win”), worse (“Loss”), or not statistically different (“Tie”) relative to a comparator using a two-sided paired t-test over the 10 random folds with a significance level of 0.05.

Hyperparameters. We tune NGBoost the same way as in Duan et al. [22]. Since PGBM, CBU, and IBUG optimize a point prediction metric, we tune their hyperparameters similarly. We also tune variance calibration parameters $\gamma$ and $\delta$ for each method (§3.2). Exact hyperparameter values evaluated and selected are in §B.2. Unless specified otherwise, we use CatBoost [52] as the base model for IBUG.
We also compare IBUG with two additional baselines—NGBoost and PGBM, winning on 17 and 11 (out of 22) datasets respectively, while losing on only 2 and 2 (respectively). Since CBU and IBUG performance is similar, we combine the two approaches, averaging their outputs; we denote this simple ensemble IBUG+CBU. Surprisingly, IBUG+CBU works very well, losing on only a maximum of 2 datasets when faced head-to-head against any other method; these results suggest IBUG and CBU are complimentary approaches.

On point performance, PGBM, CBU, and IBUG performed significantly better than NGBoost; this is consistent with previous work and is perhaps unsurprising since NGBoost is optimized for probabilistic performance, not point performance. However, IBUG generally performed better than PGBM, winning on 13 datasets and losing on only 1; and performed slightly better than CBU, winning on 6 datasets with no losses.

We also compare IBUG with two additional baselines—kNN and BART shown in §2, 3 due to space constraints. We find IBUG generally outperforms these methods in both probabilistic and point performance. Overall, the results in this section suggest IBUG generates both competitive probabilistic and point predictions compared to existing methods.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>NGBoost</th>
<th>PGBM</th>
<th>CBU</th>
<th>IBUG</th>
<th>IBUG+CBU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ames</td>
<td>38346(547)</td>
<td>10872(355)</td>
<td>11008(330)</td>
<td>10434(367)</td>
<td>10194(368)</td>
</tr>
<tr>
<td>Bike</td>
<td>12.4(0.955)</td>
<td>1.183(0.041)</td>
<td>0.833(0.036)</td>
<td>0.974(0.048)</td>
<td>0.766(0.032)</td>
</tr>
<tr>
<td>California</td>
<td>1e11(1.11)</td>
<td>0.222(0.001)</td>
<td>0.217(0.001)</td>
<td>0.213(0.001)</td>
<td>0.207(0.001)</td>
</tr>
<tr>
<td>Communities</td>
<td>0.068(0.002)</td>
<td>0.068(0.002)</td>
<td>0.067(0.002)</td>
<td>0.065(0.002)</td>
<td>0.065(0.002)</td>
</tr>
<tr>
<td>Concrete</td>
<td>3.410(0.182)</td>
<td>1.927(0.086)</td>
<td>1.788(0.077)</td>
<td>1.849(0.098)</td>
<td>1.741(0.082)</td>
</tr>
<tr>
<td>Energy</td>
<td>0.519(0.043)</td>
<td>0.147(0.006)</td>
<td>0.196(0.009)</td>
<td>0.143(0.009)</td>
<td>0.157(0.008)</td>
</tr>
<tr>
<td>Facebook</td>
<td>4.022(0.099)</td>
<td>3.554(0.095)</td>
<td>3.211(0.059)</td>
<td>3.073(0.066)</td>
<td>2.977(0.070)</td>
</tr>
<tr>
<td>Kin8nm</td>
<td>0.095(0.001)</td>
<td>0.061(0.001)</td>
<td>0.057(0.001)</td>
<td>0.051(0.001)</td>
<td>0.051(0.001)</td>
</tr>
<tr>
<td>Life</td>
<td>2.897(1.465)</td>
<td>0.815(0.027)</td>
<td>0.772(0.024)</td>
<td>0.794(0.023)</td>
<td>0.731(0.022)</td>
</tr>
<tr>
<td>MEPS</td>
<td>5.527(0.196)</td>
<td>6.448(0.092)</td>
<td>6.050(0.109)</td>
<td>6.150(0.114)</td>
<td>6.016(0.113)</td>
</tr>
<tr>
<td>MSD</td>
<td>4.524(0.005)</td>
<td>4.576(0.005)</td>
<td>4.363(0.004)</td>
<td>4.410(0.005)</td>
<td>4.347(0.004)</td>
</tr>
<tr>
<td>Naval</td>
<td>0.003(0.000)</td>
<td>0.000(0.000)</td>
<td>0.000(0.000)</td>
<td>0.000(0.000)</td>
<td>0.000(0.000)</td>
</tr>
<tr>
<td>News</td>
<td>2191(47.5)</td>
<td>2361(52.6)</td>
<td>2346(52.6)</td>
<td>2545(41.0)</td>
<td>2380(52.1)</td>
</tr>
<tr>
<td>Obesity</td>
<td>3.208(0.028)</td>
<td>1.860(0.022)</td>
<td>1.740(0.017)</td>
<td>1.866(0.021)</td>
<td>1.771(0.019)</td>
</tr>
<tr>
<td>Power</td>
<td>2.105(0.023)</td>
<td>1.531(0.019)</td>
<td>1.473(0.022)</td>
<td>1.542(0.020)</td>
<td>1.471(0.021)</td>
</tr>
<tr>
<td>Protein</td>
<td>5427(5.099)</td>
<td>1.823(0.011)</td>
<td>1.788(0.009)</td>
<td>1.784(0.008)</td>
<td>1.742(0.009)</td>
</tr>
<tr>
<td>STAR</td>
<td>132(1.589)</td>
<td>131(1.380)</td>
<td>130(1.283)</td>
<td>130(1.214)</td>
<td>129(1.198)</td>
</tr>
<tr>
<td>Superconductor</td>
<td>2.405(0.028)</td>
<td>0.126(0.004)</td>
<td>0.150(0.004)</td>
<td>0.157(0.006)</td>
<td>0.128(0.004)</td>
</tr>
<tr>
<td>Synthetic</td>
<td>5.779(0.042)</td>
<td>5.737(0.039)</td>
<td>5.739(0.040)</td>
<td>5.739(0.040)</td>
<td>5.730(0.040)</td>
</tr>
<tr>
<td>Wave</td>
<td>57120(883)</td>
<td>3891(73.9)</td>
<td>2349(10.3)</td>
<td>2679(16.0)</td>
<td>2026(9.538)</td>
</tr>
<tr>
<td>Wine</td>
<td>0.385(0.005)</td>
<td>0.323(0.005)</td>
<td>0.337(0.006)</td>
<td>0.322(0.006)</td>
<td>0.321(0.006)</td>
</tr>
<tr>
<td>Yacht</td>
<td>1.177(0.158)</td>
<td>0.292(0.042)</td>
<td>0.281(0.048)</td>
<td>0.276(0.048)</td>
<td>0.255(0.046)</td>
</tr>
</tbody>
</table>

5.2 Probabilistic and Point Performance

We first compare IBUG’s probabilistic and point predictions to each baseline on each dataset. See Table 1 for detailed CRPS results; due to space constraints, results for additional probabilistic metrics (e.g., NLL) as well as point performance results are in §B. Our main findings are as follows:

- On probabilistic performance, IBUG performs equally well or better than NGBoost and PGBM, winning on 17 and 11 (out of 22) datasets respectively, while losing on only 2 and 2 (respectively). Since CBU and IBUG performance is similar, we combine the two approaches, averaging their outputs; we denote this simple ensemble IBUG+CBU. Surprisingly, IBUG+CBU works very well, losing on only a maximum of 2 datasets when faced head-to-head against any other method; these results suggest IBUG and CBU are complimentary approaches.

- On point performance, PGBM, CBU, and IBUG performed significantly better than NGBoost; this is consistent with previous work and is perhaps unsurprising since NGBoost is optimized for probabilistic performance, not point performance. However, IBUG generally performed better than PGBM, winning on 13 datasets and losing on only 1 dataset; and performed slightly better than CBU, winning on 6 datasets with no losses.
Table 2: Probabilistic (CRPS, NLL) performance on the test set for IBUG using different base models. Results are averaged over 10 folds, and standard errors are shown in subscripted parentheses; lower is better. On 6 and 5 datasets, respectively, either IBUG-LightGBM or IBUG-XGBoost significantly outperforms IBUG-CatBoost on the validation set and subsequently on the test set, demonstrating the potential for improved probabilistic performance by using IBUG with different base models.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>CatBoost (±)</th>
<th>LightGBM (±)</th>
<th>XGBoost (±)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bike</td>
<td>0.974 (0.048)</td>
<td>0.819 (0.024)</td>
<td>0.849 (0.012)</td>
</tr>
<tr>
<td>MSD</td>
<td>4.410 (0.005)</td>
<td>4.372 (0.005)</td>
<td>4.415 (0.005)</td>
</tr>
<tr>
<td>News</td>
<td>2545 (41.0)</td>
<td>2436 (50.8)</td>
<td>2551 (56.0)</td>
</tr>
<tr>
<td>Power</td>
<td>1.542 (0.020)</td>
<td>1.536 (0.022)</td>
<td>1.515 (0.018)</td>
</tr>
<tr>
<td>Protein</td>
<td>1.784 (0.008)</td>
<td>1.683 (0.009)</td>
<td>1.785 (0.008)</td>
</tr>
<tr>
<td>Supercon.</td>
<td>0.153 (0.006)</td>
<td>0.099 (0.005)</td>
<td>0.010 (0.003)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>CatBoost (±)</th>
<th>LightGBM (±)</th>
<th>XGBoost (±)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bike</td>
<td>1.886 (0.056)</td>
<td>1.292 (0.048)</td>
<td>1.662 (0.024)</td>
</tr>
<tr>
<td>MSD</td>
<td>3.415 (0.002)</td>
<td>3.409 (0.002)</td>
<td>3.402 (0.002)</td>
</tr>
<tr>
<td>Naval</td>
<td>-6.208 (0.010)</td>
<td>-6.281 (0.007)</td>
<td>-5.853 (0.014)</td>
</tr>
<tr>
<td>Obesity</td>
<td>2.640 (0.009)</td>
<td>2.593 (0.016)</td>
<td>2.624 (0.010)</td>
</tr>
<tr>
<td>Supercon.</td>
<td>0.783 (0.181)</td>
<td>-0.496 (0.169)</td>
<td>20.4 (23.2)</td>
</tr>
</tbody>
</table>

Figure 1: *Left:* Distribution of the $k$-nearest training examples for 5 randomly-selected test instances from the MEPS (top) and Wine (bottom) datasets. *Right:* Test NLL (with standard error) when modeling the posterior using two different distributions (lower is better). IBUG can model parametric and non-parametric distributions that better fit the underlying data than assuming normality.

5.3 Different Base Models

Here we experiment using different base models for IBUG besides CatBoost [52]; specifically, we use LightGBM [36] and XGBoost [12], two popular gradient boosting frameworks. Table 2 shows that using a different base model can result in improved probabilistic performance. This highlights IBUG’s agnosticism to GBRT type, enabling practitioners to apply IBUG to future models with improved point prediction performance.

5.4 Posterior Modeling

One of the unique benefits of IBUG is the ability to directly model the output using empirical samples (Figure 1), giving practitioners a better sense of the output distribution for specific predictions. IBUG can optimize a distribution *after* training, and has more flexibility in the types of distributions it can model—from distributions using just location and scale to those with high-order moments as well as non-parametric density estimators. To test this flexibility, we model each probabilistic prediction using the following distributions: normal, skewnormal, lognormal, Laplace, student t, logistic, Gumbel, Weibull, and KDE; we then select the distribution with the best average NLL on the validation set, and evaluate its probabilistic performance on the test set.

Figure 1 demonstrates that the selected distributions for the MEPS and Wine datasets achieve better probabilistic performance than assuming normality. Qualitatively, the empirical densities of $A^{(k)}$ for a randomly sampled set of test instances reaffirms the selected distributions. As an additional comparison, we report CBU achieves a test NLL of $3.699_{\pm0.038}$ and $1.025_{\pm0.025}$ for the MEPS and wine datasets (respectively) using a normal distribution, while IBUG achieves $-6.887_{\pm0.260}$ and $0.785_{\pm0.025}$ using Weibull and KDE estimation (respectively). For the MEPS dataset, the selected Weibull distribution takes a shape parameter, which IBUG estimates directly on a *per prediction* basis using $A^{(k)}$ and MLE. In contrast, PGBM or CBU would need to optimize a global shape value using a validation set, which is likely to be suboptimal for individual predictions.
Table 3: Probabilistic performance comparison of each method with vs. without variance calibration. In all cases, calibration maintains or improves performance; it is especially helpful for CBU.

<table>
<thead>
<tr>
<th>Method</th>
<th>CRPS Wins</th>
<th>CRPS Ties</th>
<th>CRPS Losses</th>
<th>NLL Wins</th>
<th>NLL Ties</th>
<th>NLL Losses</th>
</tr>
</thead>
<tbody>
<tr>
<td>NGBoost</td>
<td>9</td>
<td>13</td>
<td>0</td>
<td>1</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>PGBM</td>
<td>13</td>
<td>9</td>
<td>0</td>
<td>11</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>CBU</td>
<td>17</td>
<td>5</td>
<td>0</td>
<td>11</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>IBUG</td>
<td>13</td>
<td>9</td>
<td>0</td>
<td>5</td>
<td>17</td>
<td>0</td>
</tr>
</tbody>
</table>

5.5 Variance Calibration

Table 3 shows probabilistic performance comparisons of each method against itself with and without variance calibration. In all cases, variance calibration either maintains or improves performance for all methods, especially CBU. Overall, these results suggest that variance calibration should be a standard procedure for probabilistic prediction, unless using a method that has particularly well-calibrated predictions to begin with. We therefore use variance calibration in all of our results.

Additionally, §C.1 shows performance results for all methods without variance calibration. Overall, we observe similar relative performance trends as when applying calibration (Table 1).

Figure 2: Runtime comparison. Left: Total train time (including tuning). Right: Average prediction time per test example. Results are shown for all datasets, averaged over 10 folds (exact values are in §B.4, Tables 12 and 13). On average, IBUG has comparable training times to PGBM and CBU, but is relatively slow for prediction.

Figure 3: Change in probabilistic (NLL) performance (top) and average prediction time (in seconds) per test example (bottom) as a function of \( \tau \) for six datasets with trees sampled first-to-last; lower is better. NGBoost, PGBM, and CBU are added for additional context. The shaded regions represent the standard error. Overall, average prediction time decreases significantly as \( \tau \) decreases while test NLL often remains relatively stable, enabling IBUG to generate probabilistic predictions with significant increased efficiency.
5.6 Sampling Trees

Figure 2 shows the runtime for each method broken down into total training time (including tuning) and prediction time per test example. On average, IBUG has similar training times to PGBM and CBU, but on some datasets, IBUG is roughly an order of magnitude faster than PGBM. For predictions, IBUG is similar to PGBM but relatively slow compared to NGBoost and CBU.

However, by sampling $\tau < T$ trees when computing the affinity vector, IBUG can significantly reduce prediction time. Figure 3 shows results when sampling trees first-to-last, which typically works best over all tree-sampling strategies (alternate sampling strategies are evaluated in §C.4). As $\tau$ decreases, we observe average prediction time decreases roughly 1–2 orders of magnitude while probabilistic performance remains relatively stable until $\tau/T$ reaches roughly 1–5%, at which point probabilistic performance sometimes starts to decrease more rapidly. Note for the Ames and Life datasets, IBUG can reach the same average prediction time as NGBoost while maintaining the same or better probabilistic performance than NGBoost, PGBM, and CBU. These results demonstrate that if speed is a concern, IBUG can approximate the affinity computation to speed up prediction times while maintaining competitive probabilistic performance.

6 Additonal Related Work

Traditional approaches to probabilistic regression include generalized additive models for location, scale, and shape (GAMLSS), which allow for a flexible choice of distribution for the target variable but are restricted to pre-specified model forms [56]. Prophet [68] also produces probabilistic estimates for generalized additive models, but has been shown to underperform as compared to more recent approaches [60, 9]. Bayesian methods [43, 30] naturally generate uncertainty estimates by integrating over the posterior; however, exact solutions are limited to simple models, and more complex models such as Bayesian Additive Regression Trees (BART) [13, 41] require computationally expensive sampling techniques (e.g., MCMC [4]) to provide approximate solutions.

Other approaches to probabilistic regression tasks include conformal predictions [61, 67, 5] which produce confidence intervals via empirical errors obtained in the past, and quantile regression [32, 43, 57]. Similar to PGBM, distributional forests (DFs) [59] estimate distributional parameters in each leaf, and average these estimates over all trees in the forest. Deep learning approaches for probabilistic regression [54, 74, 3] have increased recently, with notable approaches such as DeepAR [58] and methods based on transformer architectures [38, 39].

7 Conclusion

IBUG uses ideas from instance-based learning to enable probabilistic predictions for any GBRT point predictor. IBUG generates probabilistic predictions by using the $k$-nearest training instances to the test instance found using the structure of the trees in the ensemble. Our results on 22 regression datasets demonstrate this simple wrapper produces competitive probabilistic and point predictions to current state-of-the-art methods, most notably NGBoost [22], PGBM [64], and CBU [44]. We also show that IBUG can more flexibly model the posterior distribution of a prediction using any parametric or non-parametric density estimator. IBUG’s one limitation is relatively slow prediction time. However, we show that approximations in the search for the $k$-nearest training instances can significantly speed up prediction time; predictions are also easily parallelizable in IBUG. For future work, we plan to investigate other approximations to the affinity vector such as subsampling or even reweighting training instances, which may lead to significant speed ups of IBUG.

Acknowledgments and Disclosure of Funding

We would like to thank Zayd Hammoudeh for useful discussions and feedback and the reviewers for their constructive comments that improved this paper. This work was supported by a grant from the Air Force Research Laboratory and the Defense Advanced Research Projects Agency (DARPA)—agreement number FA8750-16-C-0166, subcontract K001892-00-S05, as well as a second grant from DARPA, agreement number HR00112090135. This work benefited from access to the University of Oregon high-performance computer, Talapas.
References


Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper’s contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See §7.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See §A.1.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]

3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See §5.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See §5.1.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See §5.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See §B.1.
   (b) Did you mention the license of the assets? [No] The license of each asset can be found by following its corresponding citation.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] See §5.
   (d) Did you discuss whether and how consent was obtained from people whose data you’re using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]