Probabilistic Forecasting: A Level-Set Approach

Hilaf Hasson  
Amazon Research  
hashilaf@amazon.com

Yuyang Wang  
Amazon Research  
yuyawang@amazon.com

Tim Januschowski  
Amazon Research  
tjnsch@amazon.com

Jan Gasthaus  
Amazon Research  
gasthaus@amazon.com

Abstract

Large-scale time series panels have become ubiquitous over the last years in areas such as retail, operational metrics, IoT, and medical domain (to name only a few). This has resulted in a need for forecasting techniques that effectively leverage all available data by learning across all time series in each panel. Among the desirable properties of forecasting techniques, being able to generate probabilistic predictions ranks among the top. In this paper, we therefore present Level Set Forecaster (LSF), a simple yet effective general approach to transform a point estimator into a probabilistic one. By recognizing the connection of our algorithm to random forests (RFs) and quantile regression forests (QRFs), we are able to prove consistency guarantees of our approach under mild assumptions on the underlying point estimator. As a byproduct, we prove the first consistency results for QRFs under the CART-splitting criterion. Empirical experiments show that our approach, equipped with tree-based models as the point estimator, rivals state-of-the-art deep learning models in terms of forecasting accuracy.

1 Introduction

Tree-based methods are known to be robust, general-purpose and high-accuracy methods for general machine learning and data science tasks, particularly those that are not image, video or text based. This is exemplified by their popularity in competitions. In a recent interview, Anthony Goldbloom, the CEO of Kaggle, called the prominence of gradient-boosted trees on Kaggle the “most glaring difference” between what is used on Kaggle and what is “fashionable in academia,” a fact also reflected in the Kaggle Data Science and Machine Learning surveys [3, 4]. In time series forecasting in particular, tree-based methods have performed solidly in public competitions over the years [10], but have had a recent boost in attention via the M5 accuracy competition [23], arguably the most influential forecasting competition, which was dominated by tree-based methods. However, this recent surge in interest is not accompanied by methodological advances. In particular, while being able to generate probabilistic predictions ranks among the top desirable properties of forecasting techniques, the main tree-based method that outputs predictions for multiple quantiles remains Quantile Regression Forests (QRFs), which does not take advantage of gradient boosted trees algorithms.

In this manuscript we provide such an advancement and show its theoretical and empirical soundness: we introduce a novel algorithm, Level Set Forecaster (LSF), which can turn any point estimator algorithm into a probabilistic one. Applied to XGBoost [11] (in which case we refer to LSF as XLSF), and with a simple processing component that turns time series data into tabular data, this yields one of the first notable methodological advancements towards creating tree-based probabilistic forecasts.

We now define \( \tilde{V} \) or Voronoi sets \( y \v \) (associated with the ascending values \( \text{Theorem 3} \) for QRFs). We discuss related work in Section 5, followed by the empirical studies for

The rest of the article is structured as follows. In Section 2 we introduce the Level Set Forecaster \( X,Y \) (partitioning algorithm is that for a test sample \( \text{arriving at the final partition} \ R \tilde{\}) \). With the basic set-up described above, the cardinality of each bin \(|\{\{y_1,...,y_k\}|x_i \in R \tilde{\} \) may be that \( A \). Forming the partition of the feature space. With a new testing data point \( x \), we find the partition cell whose predictions are “close to” \( f(x) \), and the empirical samples that belong to the same partition cell yield the predictive distribution.

1. We propose a novel algorithm, Level Set Forecaster (LSF), which can turn any point estimator algorithm into a probabilistic one. At a high level, it groups training data whose predictions are sufficiently close, and then uses the resulting bins of true values in the training set as the predicted distributions.

2. We prove the consistency of LSF (Theorem 1), and as a byproduct, prove the consistency of QRFs (Theorem 3). By building on the methods in [32], we introduce a general framework for consistency results that goes beyond random forests, and apply it to LSF and QRF as special cases.

3. We compare LSF with the state-of-the-art models in both tabular (see Section 6.1) and forecasting tasks (See Section 6.2), and the empirical results verify the effectiveness of the proposed approach.

The rest of the article is structured as follows. In Section 2 we introduce the Level Set Forecaster algorithm, and present the consistency of the resulting estimator in Section 5. We continue by describing the method of proof of the result in Section 4, as well as presenting the novel result (Theorem 3) for QRFs. We discuss related work in Section 3, followed by the empirical studies for tabular and time series data in Section 6.

2 Level Set Forecaster

Given a dataset \( D := \{(x_i,y_i)\}_{i=1,...,n} \subset \mathbb{R}^d \times \mathbb{R} \), we train a given point prediction algorithm \( A \) on \( D \) to arrive at a model \( f : \mathbb{R}^d \rightarrow \mathbb{R} \). Our goal is to output a probabilistic predictor based on \( f \), such that one could query an arbitrary quantile at any testing point \( x \in \mathbb{R}^d \). In more precise terms, assuming \( (x_i,y_i) \)'s are sampled IID from the joint distribution \( (X,Y) \), we wish to find an estimator for \( P(Y \leq y|X) \).

In Algorithm 1 we propose a conceptually simple yet effective approach to turn \( f \) into a probabilistic forecaster. At a high level, Algorithm 1 creates groupings of training data points such that their predictions are “sufficiently close.” These groupings are used to partition the feature space. With a new testing data point \( x \), we find the partition cell whose predictions are “close to” \( f(x) \), and the empirical samples that belong to the same partition cell yield the predictive distribution.

**Forming the partition of the feature space.** Algorithm 1 has a natural geometric interpretation. First, one trains the algorithm \( A \) on the data \( D \) to get a model \( f : \mathbb{R}^d \rightarrow \mathbb{R} \). The level sets \( \{f^{-1}(f(x_i))\} \) are disjoint, but they do not in general satisfy that their union covers \( \mathbb{R}^d \), because it may be that \( \{f(x_1),...,f(x_n)\} \) is not equal to the entire image of \( f \). We therefore create Voronoi sets for \( \{f(x_1),...,f(x_n)\} \), and look at inverse images of these. Namely, writing \( \{f(x_1),...,f(x_n)\} \) as \( \{v_1,...,v_k\} \) without repetition and with \( v_1 < \cdots < v_k \), the \( j^{th} \) Voronoi set is:

\[
V_j = \{a \in \mathbb{R}|v_j = \text{argmin}_{\{v_1,...,v_k\}}(|v_i - a|)\}.
\]

We now define \( \tilde{B}_j = f^{-1}(V_j) \), with each \( \tilde{B}_j \) being a disjoint union of level sets of \( f \). The inverse Voronoi sets \( B_j \)'s are disjoint, and their union covers all of \( \mathbb{R}^d \). This is not yet, however, our final partition: with the basic set-up described above, the cardinality of each bin \( \{|y_i \in \{y_1,...,y_n\}|x_i \in B_j\} \) may be too small (often equal to 1) to ensure the desired property of consistency. In the final part of the algorithm we merge the different \( \tilde{B}_j \) together, by going sequentially over \( \tilde{B}_1,\tilde{B}_2,\ldots,\tilde{B}_k \) (associated with the ascending values \( v_1,\ldots,v_k \)), grouping them until a sufficient bin size is ensured (repeated \( y_i \) values are not ignored in this computation), and then proceeding to a new grouping, arriving at the final partition \( \mathbb{R}^d = \bigcup J B_j \), where each \( B_j \) is a union of \( \tilde{B}_j \)'s. The intuition behind the partitioning algorithm is that for a test sample \( (X,Y) \) and a value \( y \in \mathbb{R} \) we expect \( P(Y \leq y|X) \) to not vary wildly as \( X \) runs over a particular partition cell \( B_j \).
We shall study the consistency property of the estimator $\hat{L}$. 

**Algorithm 1:** Level Set Partitioning Algorithm

**Input:** $D := \{(x_i, y_i)\}_{i=1,...,n} \subset \mathbb{R}^d \times \mathbb{R}$, a natural number $\text{min}_b\text{in}_\text{size}$, a point prediction algorithm $A$ (e.g., XGBoost).

Train $A$ on $D$, and call the resulting model $f : \mathbb{R}^d \rightarrow \mathbb{R}$.

Create a dictionary $\text{pred}_\text{to}_\text{bin}$ whose keys are $\{f(x_i)\}_i$, such that the value of $f(x_i)$ is the list of $y_j$ satisfying $f(x_j) = f(x_i)$.

Sort the distinct elements of $\{f(x_1), ..., f(x_n)\}$ in ascending order, and denote them $[v_1, ..., v_k]$.

Initialize an empty dictionary $\text{res}_\text{pred}_\text{to}_\text{bin}$, and an empty list $\text{current}_\text{bin}$.

**Remark 1.** Algorithm 1 resembles the classic Quantile Regression Forecasts (QRFs) (27) in the following sense. In QRFs, the random forests are trained in the regular sense, i.e., one splits the nodes according to the CART-splitting criterion. The probabilistic predictions are generated by “opening” the leaf nodes of each tree, and empirically sampling the predictions instead of predicting with the average values in the leaves. Here, the second step is exactly the same while the “leaf nodes” are replaced by partitions induced by the level sets of an arbitrary point estimator. In fact, if one used a decision tree for $A$ and let $\text{min}_b\text{in}_\text{size} = 1$ then Algorithm 1 reduces to a QRF with one tree.

**Output:** A partition such that $w_1, w_2 \in \mathbb{R}^d$ are said to be in the same partition if $\text{pred}_\text{to}_\text{bin}$ maps $\text{argmin}_{v_1, ..., v_k}(|v_i - f(w_1)|)$ and $\text{argmin}_{v_1, ..., v_k}(|v_i - f(w_2)|)$ to the same bin.

Using the partition to make inferences. In the notation of Algorithm 1 the algorithm finds the closest element $v_j$ in $[v_1, ..., v_k]$ to $f(x)$ (by binary search), and then computes and caches the quantile of $\text{pred}_\text{to}_\text{bin}[v_j]$.

In more mathematical terms, Algorithm 1 outputs a partition $\mathbb{R}^d = \bigcup B_j$ of the feature space into finitely many disjoint sets. Queried at a feature vector $x \in B_i \subset \mathbb{R}^d$, and for any value $y \in \mathbb{R}$, our probabilistic forecasting algorithm returns an estimator for $P(Y \leq y|X)$ at $X = x$:

$$\hat{\eta}_{LSF}(x) := \frac{|\{\gamma \in L_i | \gamma \leq y\}|}{|L_i|},$$

(2)

where $L_i := \{y_i \in \{y_1, ..., y_n\}|x_i \in B_i\}$ is the associated true target values for the partition cell $B_i$. We shall study the consistency property of the estimator $\hat{\eta}_{LSF}$ in the next section.

**Remark 1.** Algorithm 1 is implemented in GluonTS: https://github.com/awslabs/gluon-ts/blob/master/src/gluonts/model/rotbaum/_model.py

3 Main Theorem: LSF is Consistent

In this section, we aim to provide a quantitative understanding of the Level-set Forecaster (LSF), in particular, the consistency of LSF. The key insight that motivated the LSF algorithm was the observation that the literature regarding the consistency of Random Forests (RFs) can be extended to apply in much more general contexts.
We remark that it is precisely because the partitions of trees grown by the CART-splitting algorithm are data-dependent that consistency of RFs has only been proven recently (32), while most other work (17, 36, 6, 27, 31) has focused on easier algorithms that are not used in practice. We can leverage these methods to prove consistency of LSF as well (Theorem 1); and as a byproduct we also prove consistency of QRFs under the CART-splitting algorithm (Theorem 3).

In order to have a consistency assurance, we require mild assumptions on the data generating process, and some intuitive assumptions on the point forecasting algorithm $A$.

**Assumption 1.**

1. **Assumption on the data generating process:**
   
   (a) There is a uniformly equicontinuous family of functions $p_y(x)$ (as $y$ varies) so that $p_y(x)$ integrates to $P(Y \leq y | X)$ for all $y$. Further assume that $E(Y | X)$ is bounded.
   
   (b) For every $\varepsilon > 0$ there is a $\delta > 0$ so that for every $x$ in the image of $X$ the probability that $X$ lies in the ball of radius $\varepsilon$ centered at $x$ is at least $\delta$.
   
   (c) There exists a function $m(x)$ that integrates to $E(Y | X)$ so that $\forall \varepsilon > 0 \exists \delta > 0$ such that for all $x, x' \in \mathbb{R}^d$ and $0 \leq y \leq 1$: $|m(x) - m(x')| < \delta$ implies $|p_y(x) - p_y(x')| < \varepsilon$.

2. **Assumptions on the point estimator $A$:** Let $W_n := \{f(X_1), ..., f(X_n)\}$, with $k := |W_n|$. Assume that $A$ has a choice (for every $n$) of hyperparameters that would satisfy:
   
   (a) Collisions among the training data $f(X_i) = f(X_j)$ are rare in the precise sense that there exists a positive number $C$ such that:

   $$P \left( C < \frac{k}{n} \right) \to 1.$$

   (b) For $X$ an independent variable following the distribution of the $X_i$'s the value $f(X)$ does not tend to be extremal among the $f(X_i)$'s. To be precise, for any sequence $d_n \to 0$ of positive numbers we have that

   $$P \left( \frac{|\{v \in W_n | f(X) < v\}|}{k} > d_n \right) \to 1, P \left( \frac{|\{v \in W_n | v < f(X)\}|}{k} > d_n \right) \to 1.$$

   (c) The image of $f$ is “dense in probability”. To be precise,

   $$(\ln(n))^2 \sum_{i=2}^{k} \text{Vol}(V_i) \to 0$$

   in probability, where the $V_i$'s are the Voronoi sets from Equation 1

   (d) There's a constant $c$ so that $\forall j, \forall x \in f^{-1}(V_j) \forall \varepsilon > 0$ the set $f^{-1}(V_j) \cap B_\varepsilon(x)$ a.s. contains a ball of radius $c \cdot \varepsilon$.

   (e) The base algorithm $A$ is a mean square consistent estimator of the conditional mean:

   $$E(|f(X) - E(Y | X)|^2) \to 0.$$

**Theorem 1.** Under Assumption 1, letting $\min_{\text{bin size}} = (\ln(n))^2$, LSF is mean square consistent. That is, for any value $y \in \mathbb{R}$, we have

$$E(|\hat{y}_{LSF}(X) - P(Y \leq y | X)|^2) \to 0,$$

where the convergence is uniform in $y$.

As a byproduct of proving this result, we also give the first consistency result (Theorem 3) for QRFs grown under the CART-splitting algorithm under an additive regression model assumption.

**Remark 2.** Assumptions 1a and 1b are much weaker than the additive regression assumptions in 32. Assumption 1c is what allows the base algorithm to be informative about the conditional pdfs. In light of the empirical success of LSF (see Section 6), it appears that Assumption 1c holds quite generally in real-world data. We expect that Assumptions 1a, 1b, 1d, and 1e, which simply ensure non-degenerate behavior, hold for any reasonable choice of $A$, but the precise statement is left as a future work. (If $f$ is locally constant on hyperrectangles then Assumption 1d holds with $c = \frac{1}{\sqrt{d+1}}$.)
4 Method of Proof: Extend Results on RF Consistency to More General Settings

Our strategy for proving Theorem 1 is to generalize and strengthen existing results on the consistency of RFs, in particular through the work of [32]. We propose a unified framework that subsumes both the standard RFs (and QRFs) and the proposed LSF estimator, allowing us to arrive at the consistency results. We begin by introducing additional notation and nomenclature.

4.1 Problem Setup and Existing Results

Define a data-based partitioning algorithm \( B \) as being any algorithm taking a training dataset, i.e., a finite subset \( D = \{(X_1, Y_1), \ldots, (X_n, Y_n)\} \) of \( \mathbb{R}^d \times \mathbb{R} \) and outputting a (potentially random) partition \( \bigcup_{j=1}^M \mathbb{R}^d \) into disjoint sets \( \bigcup_{j=1}^M B_j \) satisfying that there’s a constant \( c > 0 \) so that \( \forall j \forall x \in B_j \forall \varepsilon > 0 \) the set \( B_j \cap B_\varepsilon(x) \) a.s. contains a ball of radius \( c \varepsilon \). (This holds if the \( B_j \)'s are hyperrectangles using \( c = \frac{1}{\sqrt{d+1}} \).) Examples include Algorithm 1 as well as any tree-growing algorithm such as the CART-splitting algorithm (see Algorithm 2, recalled below).

We define a mean-regression (resp. quantile-regression) \( \hat{\eta} \)-estimator trained on \( D \) as an estimator of \( \mathbb{E}(\xi(Y)|X) \), where \( \xi \) is the identity map \( \xi(Y) := Y \) (resp. \( \xi(Y) := I_{Y \leq y} \)) given by first applying \( B \) to get a partition \( \mathbb{R}^d = \bigcup_{j=1}^M B_j \); and then at inference taking a feature vector \( x \in B_l \) to: \( \frac{\sum_{\gamma \in L_l} \xi(\gamma)}{|L_l|} \), where \( L_l := \{Y_i \in \{Y_1, \ldots, Y_n\}|X_i \in B_l\} \). Now we are ready to introduce the central object of our study, generalized RF/QRF.

**Definition 1.** A Generalized RF (resp. Generalized QRF) grown using the partitioning algorithm \( B \) with \( M \) estimators is defined as:

\[
\hat{\eta}(x) := \frac{\sum_{j=1}^M \hat{\eta}_j(x)}{M},
\]

where \( \hat{\eta}_j \) is a mean-regression (resp. quantile-regression) \( B \)-estimator trained on only part of the data: for the \( j \)-th estimator subsample uniformly without replacement \( n \geq n \) many datapoints from \( D \) to train on, where \( \alpha_n \) is a (non-random) sequence of natural numbers satisfying \( 1 \leq \alpha_n \leq n \).

We remark that LSF is simply a generalized QRF grown using Algorithm 1 with one estimator \((M = 1)\) and full subsampling \((\alpha_n = n)\); and that RFs (resp. QRFs) are simply generalized RFs whose partitioning algorithm is induced by a tree-growing algorithm such as the CART splitting algorithm (see Algorithm 2).

To fix some notation, we let \( t_n \) (a random variable that can depend on the data) be the number of partitions in the first \( B \)-estimator; and we let \( R_{x,n} \) be the partition in the first \( B \)-estimator that contains a feature vector \( x \in \mathbb{R}^d \).

The main question we consider is:

**Problem 1.** Under what assumptions is the generalized RF/QRF estimator \( \hat{\eta}(\cdot) \) constructed in 4 \( L^2 \) (mean square) consistent? i.e.,

\[
\mathbb{E}(|\hat{\eta}(X) - \mathbb{E}(\xi(Y)|X)|^2) \to 0,
\]

where \( \xi \) is defined either as \( \xi(T) = T \) or \( \xi(T) = I_{T \leq y} \).

Our investigation starts with the recent work on the consistency of random forests. In Theorem 1 of [32] the authors proved mean square consistency in the mean regression case \( \xi(T) = T \) under some mild assumptions, where the partitioning algorithm is the one induced by a tree grown via the CART splitting criterion.

We briefly recall the tree-growing algorithm for the CART splitting criterion:
Algorithm 2: Mean and Quantile Regression Forests with the CART-Splitting Criterion; CART loss is recalled in Appendix A. Note that unlike Algorithm 1 the number of partition cells $t_n$ here is a hyper-parameter and therefore not random.

**Input:** We fix a natural number $m_{try} \leq d$; and sequences of natural numbers $\{t_n\}_n, \{a_n\}_n$ such that $t_n \leq a_n \leq n$.

**while not reaching $t_n$ many leaves do**

- For the subsampling of each tree, choose $a_n$ points uniformly without replacement from the training set.
- At each step, and for each leaf, choose the split that minimizes CART loss, considering only $m_{try}$ split dimensions sampled uniformly without replacement from $\{1, \ldots, d\}$.

**Mean prediction:** output the average value of the leaf nodes. $(\xi(T) = T)$.

**Quantile prediction:** output the average value of the leaf nodes. $(\xi(T) = I_{T \leq y})$

In [32], the authors make the following assumptions:

**Assumption 2** (Assumptions in [32]).

1. Subsamples and tree growth are same as in the CART-splitting algorithm.

2. Additive Regression Model Assumption: $Y$ is normal with fixed variance $\sigma^2$ and with mean $\sum_{i=1}^{d} m_i p_i(X)$, where $p_i$ is the projection onto the $i$th coordinate, where each $m_i : [0, 1] \rightarrow \mathbb{R}$ is a continuous function, and where the marginal distribution of $X$ is uniform in the hypercube $[0, 1]^d$.

3. Assume that $\lim_{n \rightarrow \infty} \frac{t_n \ln(a_n)}{a_n} = 0$, $\lim_{n \rightarrow \infty} t_n = \infty$.

The result of primary interest in [32] is the following.

**Theorem 2.** ([32], Theorem 1) Under Assumptions 2 we have mean square consistency

$$\mathbb{E}\left[|\hat{\eta}_1(X) - \mathbb{E}(Y|X)|^2\right] \rightarrow 0,$$

where $\xi(T) := T$ is the identity transformation.

**Remark 3.** The authors [32] phrase their result as the consistency of $\mathbb{E}(\hat{\eta}_1(X)|X, \{X_i, Y_i\}_i)$, i.e. the case of an infinite number of trees, but they in fact reduce first to the stronger statement that a single tree is consistent.

In [27], the quantile regression case $(\xi(T) = I_{T \leq y})$ was addressed, but only for the case of label-independent weights, which precludes the case of the partitioning algorithm being induced by trees grown via the CART-splitting algorithm.

### 4.2 Consistency of Generalized RFs/QRFs

To begin with, we state a novel result, the first consistency proof for QRFs under the CART-splitting criterion.

**Theorem 3.** Under Assumption 2 with the assumption replaced by the weaker assumption that $\lim_{n \rightarrow \infty} \frac{t_n \ln(a_n)}{a_n} = 0$, $\lim_{n \rightarrow \infty} t_n = \infty$, we have that:

$$\mathbb{E}(|\hat{\eta}_1(X) - P(Y \leq y|X)|^2) \rightarrow 0,$$

uniformly in $y$, where $\xi(T) := I_{T \leq y}$.

We remark that convergence in probability (the notion used in [27]) and $L^2$ convergence coincide for QRFs because the estimator and the CDF are both bounded.

Both Theorems 3 and Theorem 1 are corollaries from the following generalization of Theorem 3:

\[ \text{In order to be rigorous one really ought to use random closed sets. We will ignore this subtlety for convenience, and assume that the clever reader can deduce the proper adjustments.} \]
Theorem 4. A generalized QRF with a single estimator (a “generalized quantile regression tree”) is mean square consistent under the data generating assumptions $\text{Assumption 1a, 1b}$ in Assumption 1 and the following conditions:

1. The expected variance of the conditional CDFs in the cell containing $X$ goes to 0 uniformly in the quantile. To be more precise, for $(X', Y')$ an IID copy of $(X, Y)$ restricted to the sub probability space where $X' \in R_{X,n}$, assume that $\mathbb{E}(V(P(Y' \leq y|X'))|X))$ goes to 0 uniformly in $y$.

2. $\frac{t_n \ln(a_n)}{a_n} \to 0$ in probability.

Namely, under these conditions, we have that:

$$
\mathbb{E} \left[ |\hat{\theta}(X) - P(Y \leq y|X)|^2 \right] \to 0,
$$

where the convergence is uniform in $y$, and where $\xi(T) := I_{T \leq y}$.

Theorem 4 follows from Theorem 3 as an immediate corollary:

Proof. (Theorem 1) By Assumption 2a and our choice min_bin_size = $(\ln(n))^2$, it follows that in probability $t_n$ has the same order of magnitude as $\frac{n}{(\ln(n))^2}$. It is therefore clear that Assumption 2 in Theorem 4 is satisfied.

Note that $R_{X,n}$ is the union of at most $(\ln(n))^2$ many $f^{-1}(V_j)$. By Assumption 2b with probability converging to 1 none of these $V_j$’s are $V_l$ nor $V_{lb}$. Now Assumption 2c immediately implies that Vol$(f(R_{X,n}))$ goes to 0 in probability. Letting $(X', Y')$ be an IID copy of $(X, Y)$ restricted to the sub probability space where $X' \in R_{X,n}$, this implies that $\mathbb{E}(V(f(X'))|X)) \to 0$. By Assumption 2e we have that $V(f(X)) - \mathbb{E}(Y|X)) \to 0$, and thus (by the law of total variance) $\mathbb{E}(V(f(X') - \mathbb{E}(Y'|X'))|X)) \to 0$. Therefore $\mathbb{E}(V(\mathbb{E}(Y'|X'))|X)) \to 0$. By an application of Chebyshev’s Inequality and the boundedness of $\mathbb{E}(|Y|, X)$, Assumption 1c implies that $\mathbb{E}(V(P(Y' \leq y|X'))|X)) \to 0$ uniformly in $y$.

Note that Assumptions 2a, 2b and 2c could have been replaced simply by $\mathbb{E}(V(f(X'))|X)) \to 0$ in the notation of the proof above. It is less obvious to see why Theorem 4 implies Theorem 3. The assertion boils down to proving that the assumptions of Theorem 3 imply Assumption 1 of Theorem 4. We refer to Appendix B for the proof of Theorems 3 and 4.

5 Related Work

The classical conformal prediction algorithm [33], whose underlying principle is bootstrapping the residuals on a validation set, is another approach for turning point estimator algorithms into probabilistic ones. The natural setting in this approach is to create prediction intervals, rather than to make predictions for particular quantiles. Unlike LSF, the prediction intervals from conformal predictions are always symmetric about the point prediction with a fixed length. More sophisticated conformal prediction algorithms exist that allow for variable length prediction intervals (e.g. [29], which takes as input two quantile regression estimators), but they no longer satisfy that they take a single point estimator algorithm to a probabilistic one. We refer to Appendix C for more details.

The main available probabilistic tree-based algorithm other than LSF (applied to a tree-based point estimator algorithm) that does not require a list of quantiles during training is QRFs [27]. We remark that LSF applied to a decision tree is equivalent to a QRF with a single tree, and that therefore QRFs with multiple trees are an ensemble of these simple LSF algorithms.

Another approach for creating tree-based probabilistic predictions is to learn the parameters of a family of conditional distributions (e.g., [9]). For tree-based methods, this has been proposed [25, 26]. We do not further compare against this method in our experiments because the implementations are not available and instead focus on comparisons with state-of-the-art methods for forecasting that use a similar approach (e.g., DeepAR [30]).
For probabilistic time series forecasting, a number of global models have been proposed (e.g., [28, 12, 22, 14, 37]). For a comprehensive review for neural network models for forecasting, please refer to [16, 5]. In contrast to classical forecasting models (e.g., [17]) which are local (parameters are estimated per time series independently), tree-based methods are best employed as global models (learning parameters over the entire panel of time series, see e.g., [19] for a discussion and [35] for a concurrent contribution). In our empirical comparisons, we chose DeepAR [30] and CNN-QR [38] as global, deep-learning based models for their robust state-of-the-art performance.

6 Experiments

In Section 6.1 we compare XLSF (LSF with XGBoost) against QRFs (as a leading tree-based probabilistic algorithm) and Conformalized Predictions (as a baseline for turning point estimators into probabilistic ones). In Section 6.2 we apply LSF to time series data. All experiments are done using Amazon Sagemaker [21] with instance type ml.m4.16xlarge. All hyperparameters used are specified in Appendix F.

6.1 Experiments on Tabular Data

A main competitor for LSF for using tree-based methods to create probabilistic predictions is QRFs [27]. In the experiments we use the skgarden implementation [2]. QRFs are made up of multiple trees, but XLSF is one generalized tree made up of an XGBoost ensemble. It is therefore interesting to see how the two compare. By design, conformal predictions are phrased in terms of prediction intervals (a pair of quantiles) rather than a range of quantiles. For example for an \( \alpha = 0.1 \), conformal predictions aim to estimate \( P_{0.05} \) and \( P_{0.95} \) so as to have an accuracy of the prediction intervals (percent of times that the true value is in the prediction interval) that is at least 90%. We therefore chose only the pair of quantiles \( P_{0.05} \) and \( P_{0.95} \) in our experiments, as opposed to 3 quantiles as in Section 6.2.

The results are shown in Table 1 and we see that XLSF outperforms conformal predictions by a wide margin on quantile prediction, though conformal predictions’ prediction interval accuracy was more faithfully close to 90%. We remark that to reap the benefits of both algorithms, one can feed XLSF into the “Conformalized Quantile Prediction” algorithm; see [29], and the discussion in Appendix C. We also remark that QRF is much slower compared to XLSF, as shown in Table 1. The datasets in this section were small, but in Section 6.2 we will see that this makes classical QRF impractical for time series predictions, while XLSF is quite competitive.

| Dataset | XLSF | | | | QRF | | | | | | Conformalized Predictions |
|---------|------|------|------|------|------|------|------|------|------|------|
|         | P05  | P95  | accuracy (s) | | P05  | P95  | accuracy (s) | | P05  | P95  | accuracy (s) |
| facebook1 | 0.103 | 0.288 | 94.31% | 7.668 | 0.094 | 0.317 | 92.74% | 29.918 | 0.471 | 0.428 | 89.98% | 6.942 |
| facebook2 | 0.097 | 0.293 | 95.32% | 14.790 | 0.094 | 0.299 | 92.93% | 103.61 | 0.391 | 0.445 | 89.89% | 14.442 |
| meps19 | 0.100 | 0.562 | 93.37% | 5.874 | 0.111 | 0.687 | 89.45% | 9.159 | 0.633 | 0.766 | 89.64% | 5.305 |
| meps20 | 0.100 | 0.664 | 92.78% | 6.384 | 0.107 | 0.674 | 88.37% | 10.526 | 0.498 | 0.774 | 90.13% | 5.884 |
| meps21 | 0.100 | 0.555 | 92.68% | 6.215 | 0.105 | 0.635 | 88.40% | 9.111 | 0.525 | 0.802 | 89.94% | 5.246 |
| concrete | 0.036 | 0.039 | 76.69% | 0.627 | 0.036 | 0.037 | 82.52% | 0.167 | 0.031 | 0.042 | 86.89% | 0.089 |
| star | 0.011 | 0.012 | 78.52% | 1.004 | 0.011 | 0.014 | 79.21% | 0.500 | 0.011 | 0.011 | 87.99% | 0.305 |
| bio | 0.082 | 0.132 | 87.63% | 13.195 | 0.073 | 0.096 | 84.40% | 33.153 | 0.103 | 0.128 | 89.99% | 4.528 |
| community | 0.105 | 0.189 | 76.19% | 1.120 | 0.079 | 0.187 | 86.71% | 1.132 | 0.136 | 0.184 | 92.48% | 0.955 |
| bike | 0.056 | 0.059 | 87.74% | 3.403 | 0.043 | 0.044 | 80.53% | 2.828 | 0.050 | 0.048 | 91.23% | 0.734 |

Table 1: Benchmarking results: the datasets were taken from https://github.com/yromano/cqr/tree/master/datasets. Accuracy is percent of times the true value was in the prediction interval. (It should revolve around 90%.) P05 and P95 are weighted quantile losses.

6.2 LSF for Time Series Prediction

In order to apply LSF to time series data, a few choices need to be made. First, the method for preprocessing the data into tabular data; second, whether to use a single model for the entire forecast...
quantile | LSF-wrapping of winning M5 solution | Winner | Runner-up | Third | Fourth | Fifth
---|---|---|---|---|---|---
0.005 | 0.01227 | 0.010 | 0.04201 | 0.01891 | 0.01225 | 0.01133
0.025 | 0.05484 | 0.05004 | 0.08649 | 0.06446 | 0.05483 | 0.05739
0.165 | 0.31677 | 0.31276 | 0.33782 | 0.33371 | 0.31566 | 0.35329
0.25 | 0.45108 | 0.44436 | 0.46158 | 0.46699 | 0.44937 | 0.49415
0.5 | 0.72434 | 0.69886 | 0.69044 | 0.74712 | 0.71279 | 0.7661
0.75 | 0.72848 | 0.68747 | 0.72231 | 0.69522 | 0.71071 | 0.70321
0.835 | 0.60748 | 0.58519 | 0.59897 | 0.59059 | 0.61292 | 0.59058
0.975 | 0.35551 | 0.33624 | 0.35702 | 0.35378 | 0.34714 | 0.36087

Table 2: The LSF-wrapping of the winning M5 accuracy competition's solution on the top point forecast submission compared with top results in the uncertainty competition. Note that the non-monotonous nature of the metrics for the winning solutions is due to the fact that all of the numbers in this table are being evaluated only on the bottom level of the hierarchy, and that the metric used is different from the competitions’ metric.

horizon, or one model for each timestep; and finally, one has to decide on what point estimator algorithm \( A \) to use and its hyperparameters. The winning solution to the M5 accuracy competition has already made all of these decisions, and so we can simply wrap it with LSF, and see how it compares with the top 5 solutions for the M5 uncertainty competition. We also propose a simple default choice (using only lag features, one model for each timestep, and using XGBoost) and apply it to multiple datasets.

**M5 competition** The M-competitions are a series of competitions led by Spyros Makridakis intended to evaluate and compare the accuracy of different forecasting methods. The most recent competition, the M5 competition [23], was in fact a twin competitions: an accuracy competition (point forecasting) and an uncertainty competition (probabilistic forecasting) with the same training data. The winning solution (by YeonJun-IN, available in [1]) to the M5 accuracy competition is a tree-based solution that trains 220 lightgbm ([20]) models (with different feature engineering set ups and/or trained on different parts of the data) and combines them to create a point forecast. The preprocessing involved more than mere lag features.

We remark that the M5 uncertainty competition requires forecasts not only of the individual time series, but also of hierarchical aggregates. In our reported metrics we only consider the bottom level of the hierarchy, since there is no trivial way to produce aggregate forecasts using the LSF-wrapping of the winning M5 accuracy competition’s solution.

We can see that wrapping the M5 accuracy competition’s winning solution is competitive among the top 5 solutions for the M5 uncertainty competition, showing the out-of-the-box we were able to turn quality point-forecasts into quality probabilistic forecasts.

**Benchmarking datasets.** We now turn to some benchmarking with certain default choices of preprocessing and inference on common time series datasets. To be precise, we preprocess the time series data into tabular data by sampling \( P \) context windows with only lag features, and then training a separate probabilistic forecaster for each time-step in the future. (See Algorithm 3 in the appendix for more details; see also Appendix F.)

We refer to this set-up as Tree-based Time Series Wrapper (TTSW). We remark that by design, TTSW with XLSF never predicts above (resp. below) the maximal (resp. minimal) value observed. We also include TTSW with QRFs instead of XLSF, and with lightgbm ([20]) with quantile loss, which we refer to as Quantile Regression. If one chooses XLSF or QRF there is no need to specify quantiles at training time, but for Quantile Regression, one specifies quantiles before training, and for each timestep in the forecast horizon it trains as many models as the number of quantiles requested.
By default $P = 1000000$ and the context window is equal to the forecast horizon. In Table 3, since QRF is much slower to train, we had to reduce $P$ to 10000, which means it had 100 times less data-points to train on. Even with this adjustment, it took considerably longer than the other methods. Our results in Table 3 show that TTSW (both XLSF and with Quantile Regression) perform competitively with state-of-the-art deep learning forecasting methods ([30, 38]), while maintaining advantages of tree-based methods such as interpretability. For detailed results, please refer to Appendix E.
References


A  CART Loss

Let $\tau_{i,j}$ be the indicator variable for whether $X_i$ is chosen for the $j^{th}$ estimator. We recall that CART loss for the a cell $A$ along dimension $k$ for the position cut $z$ in tree $j$ is:

$$
\frac{1}{\sum_i I_{X_i \in A \tau_{i,j}}} \sum_i (Y_i - \bar{Y}_A)^2 I_{X_i \in A \tau_{i,j}} - \frac{1}{\sum_i I_{X_i \in A \tau_{i,j}}} \sum_i (Y_i - \bar{Y}_{AL} I_{X_i^{(k)} < z} - \bar{Y}_{AR} I_{X_i^{(k)} \geq z})^2 I_{X_i \in A \tau_{i,j}},
$$

where the superscript $(k)$ denotes the $k$-th coordinate, and where

$$
\bar{Y}_A = \frac{\sum_i Y_i I_{X_i \in A \tau_{i,j}}}{\sum_i I_{X_i \in A \tau_{i,j}}}, \quad \bar{Y}_{AL} = \frac{\sum_i Y_i I_{X_i \in A \tau_{i,j}} I_{X_i^{(k)} < z}}{\sum_i I_{X_i \in A \tau_{i,j}} I_{X_i^{(k)} < z}}, \quad \text{and} \quad \bar{Y}_{AR} = \frac{\sum_i Y_i I_{X_i \in A \tau_{i,j}} I_{X_i^{(k)} \geq z}}{\sum_i I_{X_i \in A \tau_{i,j}} I_{X_i^{(k)} \geq z}}.
$$

If any of the denominators are 0, by convention, the CART loss is $\frac{1}{\sum_i I_{X_i \in A \tau_{i,j}}} \sum_i (Y_i - \bar{Y}_A)^2 I_{X_i \in A \tau_{i,j}}$.

B  Proofs

B.1  Proof of Theorem 3

Recall that cuts under the CART splitting criterion are the cuts given by Algorithm 2, which uses the CART loss given by Equation 5. In [32] theoretical cuts are defined as the cuts that follow Algorithm 2 but with the CART loss replaced by a “theoretical loss”. The theoretical loss, in turn, is defined for a cell $A$ along dimension $k$ for the position cut $z$ as:

$$
L^*(k, z) := V(Y|X \in A) - P(X^{(k)} < z|X \in A)V(Y|X^{(k)} < z, X \in A)
- P(X^{(k)} \geq z|X \in A)V(Y|X^{(k)} \geq z, X \in A)
$$

The reason for the name “theoretical cuts” is that by the strong law of large numbers CART loss converges almost surely to the theoretical loss, which is not dependent on the data.

We first remark that in [32] their proof of Lemma 1 in fact proves more than its statement:

**Lemma 1.** Under Assumption 2, let $x \in \mathbb{R}^d$, let $I_k$ be the hyperrectangle containing $x$ after $k$ theoretical cuts, and let $D_x := \cap_k I_k$. Then with probability 1 we have that $E(Y|X)$ is almost surely constant over $D_x$.

**Proof.** This is immediate from the proof of Lemma 1 in [32]. (Note that the randomness of $D_x$ in the statement comes from the choice of splitting dimensions, governed by $\theta$.)

As Assumptions 1a, 1b, and 1c are trivially satisfied under the additive regression model assumption, similarly to the proof of Theorem 1 it suffices to show that $E(V(E(Y'|X'|X)) \to 0$, where $(X', Y')$ is an IID copy of $(X, Y)$ restricted to the sub probability space where $X' \in R_{X,n}$. This is an immediate corollary of the following lemma:

**Lemma 2.** Under Assumption 2, let $D_X := \cap_k I_k$, where $I_k$ is the hyperrectangle containing $X$ after $k$ theoretical cuts. Then the volume $\text{Vol}(R_{X,n} \setminus D_X)$ converges to 0 in probability.

**Proof.** Fix $\varepsilon, \delta > 0$. As in [32], let $A_{k,n}(X, \theta)$ be the cell containing $X$ after only $k$ cuts, and let $A^*_k(X, \theta)$ be the cell containing $X$ after $k$ theoretical cuts. By definition, there exists a $k_0$ so that $P(\text{Vol}(A^*_{k_0}(X, \theta) \setminus D_X) > \frac{\delta}{2}) < \frac{\delta}{2}$. By Lemma 3 in [32], the cuts up to the $k_0^{th}$ step are arbitrarily close to the theoretical cuts with as high a probability as we want. In particular, for any $n$ big enough, we have that

$$P(\text{Vol}(A_{k_0,n}(X, \theta) \setminus A^*_{k_0}(X, \theta)) > \frac{\varepsilon}{2}) < \frac{\delta}{2}.
$$

As one can observe from a simple Venn diagram,

$$\text{Vol}(A_{k_0,n}(X, \theta) \setminus D_X) \leq \text{Vol}(A_{k_0,n}(X, \theta) \setminus A^*_{k_0}(X, \theta)) + \text{Vol}(A^*_{k_0}(X, \theta) \setminus D_X).$$
Further, if the approximation and estimation errors converge uniformly in

We will now show that under the assumptions of Theorem 4 the conditions of Theorem 5 hold for

We will now recast Theorem 3 from \[32\] \(\text{see also}\ [15]\), which was originally about mean regression.

We follow the same general outline as \[32\], with some notable exceptions. There they used the

As all of these quantities are bounded, it suffices to show that

\[
\lim_{n \to \infty} \mathbb{E} \left[ \inf_{g \in \mathcal{F}_n, ||g||_{\infty} \leq \beta_n} \mathbb{E}_X ((g(X) - P(Y \leq y|X))^2) \right] = 0.
\]

Finally, since the volume only gets smaller for more cuts than \(k_0\), the result follows. \(\square\)

B.2 Proof of the Main Theorem (Theorem 4)

We follow the same general outline as \[32\], with some notable exceptions. There they used the

methods in \[15\], which provides tools for proving consistency results for estimators that minimize

mean square loss among a data-dependent classes of functions. Note that both mean regression

and QRFs fall into this category.

Let \(\mathcal{F}_n\) be the set of cell-wise constant functions on the (data dependent) partition given by the

generalized tree from Theorem 4; and let \(\mathcal{I}_n\) be the (random) set of indices of the subsample chosen.

We will now recast Theorem 3 from \[32\] \(\text{see also}\ [15]\), which was originally about mean regression,

to the situation of using a generalized random forest estimator for quantile regression: (As \(P(Y \leq y|X)\) is bounded, we may omit the truncation operators appearing in the original statements.)

Theorem 5. Let \(\mathcal{F}_n, \mathcal{I}_n\) be as above. Assume that there is a sequence of real numbers \(\beta_n\) such that:

1. \(\lim_{n \to \infty} \beta_n = \infty\).

2. The approximation error goes to 0:

\[
\lim_{n \to \infty} \mathbb{E} \left[ \inf_{g \in \mathcal{F}_n, ||g||_{\infty} \leq \beta_n} \mathbb{E}_X ((g(X) - P(Y \leq y|X))^2) \right] = 0.
\]

3. The estimation error goes to 0, namely for all \(L > 0\):

\[
\lim_{n \to \infty} \mathbb{E} \left[ \sup_{g \in \mathcal{F}_n, ||g||_{\infty} \leq \beta_n} \left| \frac{1}{a_n} \sum_{i \in \mathcal{I}_n} (g(x_i) - T_L I_{y_i \leq y})^2 - \mathbb{E}((g(X) - T_L I_{Y \leq y})^2) \right| \right] = 0,
\]

where \(T_L(u) := \text{sign}(u) \min(|u|, L)\).

Then

\[
\lim_{n \to \infty} \mathbb{E}(|\hat{\eta}_1(X) - P(Y \leq y|X)|^2) = 0.
\]

Further, if the approximation and estimation errors converge uniformly in \(y\), then so does \(\mathbb{E}(|\hat{\eta}_1(X) - P(Y \leq y|X)|^2)\).

We will now show that under the assumptions of Theorem 4 the conditions of Theorem 5 hold for

\(\beta_n = (\ln(a_n)/\ln(a_n))^4\); and so Theorem 4 would follow.

B.2.1 The approximation error goes to 0 (uniformly in \(y\))

Let \(h(X) = \mathbb{E}(P(Y' \leq y|X')|X) \in \mathcal{F}_n\), where \((X', Y')\) is an IID copy of \((X, Y)\) restricted to the

sub probability space where \(X' \in R_{X,n}\). We see that

\[
\mathbb{E} \left[ \inf_{g \in \mathcal{F}_n, ||g||_{\infty} \leq \beta_n} \mathbb{E}_X ((g(X) - P(Y \leq y|X))^2) \right] \leq \mathbb{E}(h(X) - P(Y \leq y|X))^2)
\]

As all of these quantities are bounded, it suffices to show that \(|h(X) - P(Y \leq y|X)|\) goes to

0 in probability. Assume by contradiction that there exists an \(\varepsilon > 0\) for which there exists an

\(\varepsilon' > 0\) satisfying \(P(|P(Y \leq y|X') - h(X)| > \varepsilon) > \varepsilon'\) for all \(n\) big enough. Assumption \(1b\) in

particular implies that for every \(\varepsilon'' > 0\) there exists a \(\delta > 0\) so that \(P(X' \in B_{\varepsilon''}(X)|X) > \delta\).

\((P(X' \in B_{\varepsilon''}(X)|X) = P(X'' \in B_{\varepsilon''}(X)|X, X'' \in R_{X,n})\) for an IID copy \(X''\) of \(X\), which in
The classical conformalized prediction algorithm transforms a point prediction algorithm into an algorithm that outputs prediction intervals. To be precise, given a confidence level \( \epsilon \), it outputs an interval \( C(X) \) for the estimation error to go to 0. We refer to [13] for the detailed proof. Following the steps of the remainder of the argument in [32], we can show that the number of cells \( t_n \) is not a random variable, which makes the computation there somewhat more straightforward.

**The estimation error goes to 0 (uniformly in \( y \))**

This follows the same pattern as in the proof of Theorem 1 in [32], except that in our case we can be more lax about our choice of \( \beta_n \), because we don’t have to concern ourselves with the untruncated situation.

Briefly, as in [32], by Theorems 9.1 and 9.4, and Lemma 13.1 in [15], as well as Assumption 2 in [4], we have that for all \( \epsilon > 0, \epsilon' > 0 \), and \( L > 0 \):

\[
P \left( \sup_{g \in \mathcal{F}_n} \left| \sum_{i \in I_n, a} \frac{1}{a} \left( g(X_i) - T_L I_{Y_i \leq y} \right)^2 - \mathbb{E}((g(X) - T_L I_{Y \leq y})^2) \right| > \epsilon \right) 
\leq \mathbb{E}(8 \exp \left( -\frac{\epsilon^2}{2048} \right) t_n \frac{\ln(a_n)}{a_n} < \epsilon' \right) + d_{\epsilon',n};
\]

where \( C_n = \frac{\epsilon^2}{2048} - \frac{\beta_n^4 t_n \ln(a_n)}{a_n} - \frac{2\beta_n^4 t_n \ln(\frac{333\epsilon^2}{\beta_n^2})}{a_n} \).

\( d_{\epsilon',n} \) is some sequence of non-negative numbers depending on \( \epsilon' \) satisfying that \( \lim_{n \to \infty} d_{\epsilon',n} = 0 \), and where the randomness comes from the randomness of \( t_n \). By our choice of \( \beta_n \) and the use of L'Hôpital's rule, we get that \( C_n \) converges to \( \frac{\epsilon^2}{2048} \) (uniformly in \( y \), of course) in probability as \( \epsilon' \) goes to 0. To be precise:

\[
\forall \epsilon'' > 0 \lim_{\epsilon' \to 0} P \left( \left| C_n - \frac{\epsilon^2}{2048} \right| < \epsilon'' \right) = 0.
\]

**Conformalized Predictions**

In what follows, let \((X_1, Y_1), \ldots, (X_n, Y_n) \in \mathbb{R}^d \times \mathbb{R}\) be the training set, and \((X_{n+1}, Y_{n+1})\) be a feature vector being queried at inference and its associated true value.

The classical conformalized prediction algorithm transforms a point prediction algorithm into an algorithm that outputs prediction intervals. To be precise, given a confidence level \( \alpha \), we say that an algorithm is an ideal prediction interval forecaster if it takes \((X_1, Y_1), \ldots, (X_n, Y_n), X_{n+1}\) and outputs an interval \( C \) satisfying that

\[
P(Y_{n+1} \in C | X_{n+1}) = 1 - \alpha.
\]
The conformalized prediction algorithm relaxes this requirement in two ways. First, it relaxes the equality to an inequality, and secondly it asks that the inequality would hold without conditioning (which roughly translates to the inequality holding on average on $X_{n+1}$):

$$P(Y_{n+1} \in C) \geq 1 - \alpha$$

It is this last condition that is guaranteed under exchangability assumptions on the data, and no assurances are made on conditional coverage.

The conformalized predictions algorithm is defined as follows: split the training data into two $I_1$ and $I_2$. Train a mean-regression point forecasting algorithm on $I_1$, and let $\hat{\mu}$ be the resulting model.

The conformalized prediction interval for a new datapoint $X_{n+1}$ is then defined as the bootstrapping interval:

$$C := [\hat{\mu}(X_{n+1}) - Q_{1-\alpha}(I_2), \hat{\mu}(X_{n+1}) + Q_{1-\alpha}(I_2)],$$

where

$$Q_{1-\alpha}(I_2) := (1 - \alpha)(1 + 1/|I_2|)\text{-th empirical quantile of } \{|Y - \hat{\mu}(X_i)| \mid i \in I_2\}.$$

We remark while this classical conformalized predictions algorithm behaves similarly to LSF in that it takes a point forecasting algorithm and output a probabilistic forecasting algorithm, the basic idea behind conformalized predictions is fundamentally different. Namely, conformalized predictions begin with the premise that an ideal prediction interval forecaster is too much to ask for, and instead of directly optimizing for it, it attempts to optimize instead the much weaker unconditional coverage requirement. Some algorithms, such as Conformalized Quantile Regression ([29]), attempt to straddle the two approaches: it takes two quantile regression point forecasters (rather than a single mean-regression forecaster), and then adjusts them to optimize for the unconditional coverage requirement, perhaps arriving at some regularizing effect. But they no longer serve as algorithms that turns point forecasting algorithms into probabilistic forecasting algorithms.

### D Tree-based Time Series Wrapper (TTSW)

TTSW is implemented in GluonTS under the name Rotbaum: [https://github.com/awslabs/gluon-ts/blob/master/src/gluonts/model/rotbaum/](https://github.com/awslabs/gluon-ts/blob/master/src/gluonts/model/rotbaum/)

**Algorithm 3:** Tree-based Time Series Wrapper (TTSW)

**Input:** Datasets made of multiple time series, where we denote the $i^{th}$ time series by $\{Z_{i,t}\}_t$; a context window size $h$; a forecast horizon $l$; and number of context windows to sample $P$.

**train_data**, **target_data**, **model_list** = [], [], []

for $i$ in $[1, ..., P]$ do
  Choose a time series $i$ and a beginning point within the time series $t_0$ uniformly. Add the context window $Z_{i,t_0}, Z_{i,t_0+1}, ..., Z_{i,t_0+h-1}$ to **train_data**. Add $Z_{i,t_0+b}, Z_{i,t_0+1}, ..., Z_{i,t_0+b+l-1}$ to **target_data**.

for $i$ in $[1, ..., l]$ do
  Train a model (XLSF/Quantile Regression/QRF) with training data **train_data** and target data **target_data**[:, $i$], and add it to **model_list**.

**Inference:** Given a new time series for inference $Z_1, ..., Z_k$, make inferences for the feature vector $(Z_{k-h+1}, ..., Z_k)$ from all of the models in **model_list**.
E  Detailed TTSW Results

Table 4: Time Series Benchmarking

<table>
<thead>
<tr>
<th></th>
<th>TTSW (XLSF)</th>
<th>TTSW (QuantileReg)</th>
<th>TTSW (QRF)</th>
<th>DeepAR</th>
<th>CNN-QR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P10</td>
<td>P50</td>
<td>P90</td>
<td>P10</td>
<td>P50</td>
</tr>
<tr>
<td>electricity</td>
<td>0.0395</td>
<td>0.0726</td>
<td>0.0375</td>
<td>0.1203</td>
<td>0.2399</td>
</tr>
<tr>
<td>parts</td>
<td>0.4371</td>
<td>1.3362</td>
<td>1.1396</td>
<td>0.2</td>
<td>1.017</td>
</tr>
<tr>
<td>m4_daily</td>
<td>0.013</td>
<td>0.0204</td>
<td>0.0112</td>
<td>0.0385</td>
<td>0.0281</td>
</tr>
<tr>
<td>traffic</td>
<td>0.0797</td>
<td>0.1682</td>
<td>0.1758</td>
<td>0.0799</td>
<td>0.1406</td>
</tr>
<tr>
<td>wiki10k</td>
<td>0.2331</td>
<td>0.3573</td>
<td>0.2874</td>
<td>0.1696</td>
<td>0.3127</td>
</tr>
<tr>
<td>dcrideshare</td>
<td>0.2028</td>
<td>0.5289</td>
<td>0.2829</td>
<td>0.1634</td>
<td>0.4887</td>
</tr>
</tbody>
</table>

F   Choices of Hyperparameters

For the purpose of using Algorithm 3 for applying XLSF to time-series data, we always use min_bin_size = 100, and the following hyperparameters for the underlying XGBoost model:

\[
\text{XGBModel}(\text{base score}=\text{None}, \text{booster}=\text{None}, \text{colsample by level}=\text{None}, \\
\text{colsample by node}=\text{None}, \text{colsample by tree}=\text{None}, \text{gamma}=\text{None}, \text{gpu id}=\text{None}, \\
\text{importance type}=\text{’gain’}, \text{interaction constraints}=\text{None}, \\
\text{learning rate}=\text{None}, \text{max delta step}=\text{None}, \text{max depth}=5, \\
\text{min child weight}=\text{None}, \text{missing}=\text{nan}, \text{monotone constraints}=\text{None}, \\
\text{n estimators}=100, \text{n jobs}=-1, \text{num parallel tree}=\text{None}, \\
\text{objective}=\text{’reg:squarederror’}, \text{random state}=\text{None}, \text{reg alpha}=\text{None}, \\
\text{reg lambda}=\text{None}, \text{scale pos weight}=\text{None}, \text{subsample}=\text{None}, \\
\text{tree method}=\text{None}, \text{validate parameters}=\text{None}, \text{verbosity}=1)
\]

We use a context length \( h \) equal to the forecast horizon \( l \), and we set the number of context windows \( P \) to equal 1000000, unless the dataset is small, in which case it automatically chooses less. Since QRF is much slower to train, we had to reduce \( P \) to 10000, which means it had 100 times less data-points to train on. Even with this adjustment, it took considerably longer than the other methods.

In Section 6.1, we use min_bin_size = 100, and (since the resulting datasets are much smaller than the ones used in time series prediction) the following hyperparameters for the underlying XGBoost model:

\[
\text{XGBModel}(\text{base score}=0.5, \text{booster}=\text{’gbtree’}, \text{colsample by level}=1, \\
\text{colsample by node}=1, \text{colsample by tree}=1, \text{gamma}=0, \text{gpu id}=1, \\
\text{importance type}=\text{’gain’}, \text{interaction constraints}=’’, \\
\text{learning rate}=0.30000000000000004, \text{max delta step}=0, \text{max depth}=2, \\
\text{min child weight}=1, \text{monotone constraints}=’()’, \\
\text{n estimators}=100, \text{n jobs}=-1, \text{num parallel tree}=1, \\
\text{objective}=\text{’reg:squarederror’}, \text{random state}=0, \text{reg alpha}=0, \\
\text{reg lambda}=1, \text{scale pos weight}=1, \text{subsample}=1, \text{tree method}=’exact’, \\
\text{validate parameters}=1, \text{verbosity}=1)
\]

For the purpose of the M5 competition, we use min_bin_size = 200, since we query half-percent quantiles.

For both QRFs and lightgbm quantile regression we use default hyperparameters:

\[
\text{RandomForestQuantileRegressor}(\text{bootstrap}=\text{True}, \text{criterion}=\text{’mse’}, \text{max depth}=\text{None}, \\
\text{max features}=\text{’auto’}, \text{max leaf nodes}=\text{None}, \\
\text{min samples leaf}=1, \text{min samples split}=2, \\
\text{min weight fraction leaf}=0.0, \text{n estimators}=10, \\
\text{n jobs}=1, \text{oob score}=\text{False}, \text{random state}=\text{None}, \\
\text{verbose}=0, \text{warm start}=\text{False})
\]
G Detailed Tabular Data Experiments

We have re-run the tabular experiments from Section 6 five times to get confidence intervals. Note that XLSF is deterministic because the hyperparameters we chose for XGBoost (see Appendix F) preclude subsampling.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Dataset</th>
<th>P05</th>
<th>P95</th>
<th>accuracy</th>
<th>time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>XLSF</td>
<td>facebook1</td>
<td>0.103+/-0.00</td>
<td>0.288+/-0.00</td>
<td>94.31%+/-0.00%</td>
<td>7.801+/-0.022</td>
</tr>
<tr>
<td></td>
<td>facebook2</td>
<td>0.097+/-0.00</td>
<td>0.294+/-0.00</td>
<td>95.327%+/-0.00%</td>
<td>14.883+/-0.051</td>
</tr>
<tr>
<td></td>
<td>meps19</td>
<td>0.101+/-0.00</td>
<td>0.563+/-0.00</td>
<td>93.38%+/-0.00%</td>
<td>6.285+/-0.335</td>
</tr>
<tr>
<td></td>
<td>meps20</td>
<td>0.101+/-0.00</td>
<td>0.664+/-0.00</td>
<td>92.79%+/-0.00%</td>
<td>5.926+/-0.037</td>
</tr>
<tr>
<td></td>
<td>meps21</td>
<td>0.101+/-0.00</td>
<td>0.556+/-0.00</td>
<td>92.688%+/-0.00%</td>
<td>6.386+/-0.132</td>
</tr>
<tr>
<td></td>
<td>concrete</td>
<td>0.037+/-0.00</td>
<td>0.039+/-0.00</td>
<td>76.699%+/-0.00%</td>
<td>0.712+/-0.005</td>
</tr>
<tr>
<td></td>
<td>star</td>
<td>0.012+/-0.00</td>
<td>0.012+/-0.00</td>
<td>87.634%+/-0.00%</td>
<td>1.108+/-0.001</td>
</tr>
<tr>
<td></td>
<td>bio</td>
<td>0.082+/-0.00</td>
<td>0.132+/-0.00</td>
<td>95.327%+/-0.00%</td>
<td>13.648+/-0.295</td>
</tr>
<tr>
<td></td>
<td>community</td>
<td>0.106+/-0.00</td>
<td>0.19+/-0.00</td>
<td>76.19%+/-0.00%</td>
<td>1.186+/-0.011</td>
</tr>
<tr>
<td></td>
<td>bike</td>
<td>0.057+/-0.00</td>
<td>0.06+/-0.00</td>
<td>87.741%+/-0.00%</td>
<td>3.452+/-0.020</td>
</tr>
<tr>
<td>QRF</td>
<td>facebook1</td>
<td>0.103+/-0.005</td>
<td>0.319+/-0.013</td>
<td>92.383%+/-0.203%</td>
<td>30.368+/-0.121</td>
</tr>
<tr>
<td></td>
<td>facebook2</td>
<td>0.094+/-0.001</td>
<td>0.289+/-0.006</td>
<td>92.918%+/-0.147%</td>
<td>104.485+/-2.923</td>
</tr>
<tr>
<td></td>
<td>meps19</td>
<td>0.11+/-0.002</td>
<td>0.727+/-0.018</td>
<td>89.914%+/-0.516%</td>
<td>9.798+/-0.314</td>
</tr>
<tr>
<td></td>
<td>meps20</td>
<td>0.107+/-0.001</td>
<td>0.687+/-0.028</td>
<td>88.663%+/-0.331%</td>
<td>10.623+/-0.043</td>
</tr>
<tr>
<td></td>
<td>meps21</td>
<td>0.108+/-0.001</td>
<td>0.655+/-0.024</td>
<td>88.57%+/-0.371%</td>
<td>9.246+/-0.095</td>
</tr>
<tr>
<td></td>
<td>concrete</td>
<td>0.036+/-0.002</td>
<td>0.037+/-0.001</td>
<td>80.194%+/-2.109%</td>
<td>0.175+/-0.001</td>
</tr>
<tr>
<td></td>
<td>star</td>
<td>0.013+/-0.00</td>
<td>0.014+/-0.001</td>
<td>79.723%+/-0.853%</td>
<td>0.518+/-0.002</td>
</tr>
<tr>
<td></td>
<td>bio</td>
<td>0.072+/-0.00</td>
<td>0.098+/-0.001</td>
<td>84.682%+/-0.274%</td>
<td>33.258+/-0.283</td>
</tr>
<tr>
<td></td>
<td>community</td>
<td>0.088+/-0.005</td>
<td>0.168+/-0.007</td>
<td>87.569%+/-1.125%</td>
<td>1.14+/-0.008</td>
</tr>
<tr>
<td></td>
<td>bike</td>
<td>0.044+/-0.0001</td>
<td>0.046+/-0.001</td>
<td>80.358%+/-0.461%</td>
<td>2.842+/-0.036</td>
</tr>
<tr>
<td>CQR</td>
<td>facebook1</td>
<td>0.452+/-0.011</td>
<td>0.425+/-0.005</td>
<td>90.147%+/-0.151%</td>
<td>7.028+/-0.049</td>
</tr>
<tr>
<td></td>
<td>facebook2</td>
<td>0.43+/-0.019</td>
<td>0.435+/-0.007</td>
<td>90.22%+/-0.162%</td>
<td>14.552+/-0.077</td>
</tr>
<tr>
<td></td>
<td>meps19</td>
<td>0.657+/-0.039</td>
<td>0.76+/-0.011</td>
<td>90.459%+/-0.455%</td>
<td>5.423+/-0.109</td>
</tr>
<tr>
<td></td>
<td>meps20</td>
<td>0.488+/-0.026</td>
<td>0.774+/-0.014</td>
<td>90.208%+/-0.166%</td>
<td>5.926+/-0.037</td>
</tr>
<tr>
<td></td>
<td>meps21</td>
<td>0.504+/-0.034</td>
<td>0.8+/-0.007</td>
<td>90.134%+/-0.711%</td>
<td>5.288+/-0.039</td>
</tr>
<tr>
<td></td>
<td>concrete</td>
<td>0.034+/-0.002</td>
<td>0.038+/-0.002</td>
<td>87.767%+/-3.482%</td>
<td>0.091+/-0.0</td>
</tr>
<tr>
<td></td>
<td>star</td>
<td>0.011+/-0.00</td>
<td>0.012+/-0.00</td>
<td>88.037%+/-0.717%</td>
<td>0.309+/-0.002</td>
</tr>
<tr>
<td></td>
<td>bio</td>
<td>0.104+/-0.001</td>
<td>0.13+/-0.001</td>
<td>89.672%+/-0.309%</td>
<td>4.561+/-0.039</td>
</tr>
<tr>
<td></td>
<td>community</td>
<td>0.134+/-0.003</td>
<td>0.181+/-0.006</td>
<td>92.632%+/-1.192%</td>
<td>0.959+/-0.007</td>
</tr>
<tr>
<td></td>
<td>bike</td>
<td>0.05+/-0.001</td>
<td>0.049+/-0.00</td>
<td>90.101%+/-0.783%</td>
<td>0.737+/-0.003</td>
</tr>
</tbody>
</table>

Table 5: Benchmarking results: the datasets were taken from https://github.com/yromano/cqr/tree/master/datasets. Accuracy is percent of times the true value was in the prediction interval. (It should revolve around 90%.) P05 and P95 are weighted quantile losses. The experiments were run 5 times.
H Formula for Weighted Quantile Loss

The formula for weighted quantile loss for quantile $\tau$, for real values $y_i$, and predictions $q_i$:

$$wQL^{Reg}[\tau] := \frac{2 \sum \tau \max(y_i - q_i, 0) + (1 - \tau) \max(q_i - y_i, 0)}{\sum_i |y_i|}$$

In the time series situation, if $y_{i,t}$ as $t$ varies are the real values for time series number $i$ in the dataset, and $q_{i,t}$ is prediction for $y_{i,t}$ then the formula is:

$$wQL^{TS}[\tau] := \frac{2 \sum_{i,t} \tau \max(y_{i,t} - q_{i,t}, 0) + (1 - \tau) \max(q_{i,t} - y_{i,t}, 0)}{\sum_{i,t} |y_{i,t}|}$$

I LSF with Non-Tabular Data

One of the assumptions of LSF is that the base point-forecasting algorithm is tabular, and so that raises the question of how to integrate LSF with neural networks dealing with non-tabular data. To that end, LSF can come in at the embedding level of the architecture. To be a little more explicit, if $f_1$ is the portion of the neural network that embeds the data into a fixed dimensional space, and the remainder of the network is $f_2$, then after training you would cache pairs of embeddings and true target values, and later feed them into LSF together with $f_2$ as the base algorithm. At inference, rather than feeding in the raw data, you would feed the data into $f_1$ to obtain a fixed length feature vector, and only then feed it into the LSF.