Research Prelim Report: Quantifying Uncertainty in Online Regression Forests

Brian Burns

Department of Statistics, University of Washington Seattle, WA, 98195, USA

Abstract

In an online streaming setting with unbounded data and tight memory and compute requirements, we examine meta-algorithms that augment regression forests to produce confidence intervals rather than just predictions. The authors modify two batch meta-algorithms of this sort to produce variants online-suitable variants OnlineCP and OnlineQRF.

We compare these meta-algorithms to Mondrian Forests, the state-of-the-art online regression forest algorithm capable of computing confidence intervals. With respect to resource constraints, OnlineCP and OnlineQRF exhibit $O(1)$ training and prediction compute time and $O(1)$ memory cost in excess of their base learner, while Mondrian Forests take $O(\log(n))$ time to process the n^{th} sample in a datastream. In addition to their resource efficiency, we show that OnlineCP and OnlineQRF outperform Mondrian Forests in terms of error rate on a variety of small-scale datasets. We also examine these algorithms' performance under concept drift.

1 Introduction

In this report and in this open-source repository, we reproduce the work of Vasiloudis, Morales and Boström in their paper Quantifying Uncertainty in Online Regression Forests [Vasiloudis et al., 2019]. We begin by defining the problem of online confidence interval generation for regression forests, and describe two batch methods that the authors later adapt to the online setting.

1.1 Problem definition

We are given a *data stream* $(x_i, y_i)_{i=1}^n$, with $x_i \in \mathbb{R}^d$ the data points and $y_i \in \mathbb{R}$ the labels, sampled from some unknown underlying distribution π.

Unlike the batch setting in machine learning, in which we receive the whole dataset X, Y prior to model training and evaluation, in the *online setting* we receive each datapoint in the stream one-at-a-time. Moreover in the online setting, the size n of the datastream is very large or infinite, prohibiting us from storing all of the data we have received thus far. Therefore training and model evaluation must occur incrementally. The standard framework for training and evaluating a model $\hat{f}$ online is *prequential evaluation* [Gama et al., 2009], in which we, in order:

1. Receive a datapoint (x, y) from the stream.
2. Make a prediction $\hat{f}(x)$.
3. Reveal the true label y to our model.
4. Update our performance metrics (e.g. the MSE) given $x, f(x)$, and y.
5. Train our model on the datapoint (x, y).

The model is then trained gradually over the course of streaming, and model evaluation occurs via trailing performance metrics over the course of streaming.

In many online settings it is critical to have a measure of uncertainty of $\hat{f}$. The authors provide an excellent example: if $\hat{f}$ estimates the distance of a vehicle ahead in an autonomous vehicle (AV), a point estimate is insufficient for model deployment. If the AV takes 100 meters to stop, and our point prediction $\hat{f}(x)$ is 110 meters but the true label y is 90 meters, the AV will not be able to stop before crashing into the vehicle. Therefore in this case, in addition to the point prediction we must generate *confidence intervals* $[a, b]$ such that the vehicle is $y \in [a, b]$ distance ahead with probability $> 99.99\%$.

Specifically, we are interested in *augmenting* existing regression forest algorithms with given training and prediction functions to produce confidence intervals instead of mere point predictions. Therefore the aim of the investigation is to develop *meta-algorithms* which take as input a *base learner* regression forest point-prediction and training algorithm, and output a regression forest confidence-interval-prediction and training algorithm. We do so in the online setting, and therefore aim to add no more than $O(1)$ model memory, prediction time, and training time on top of whatever costs these are for the base learner.

We can now state the problem more precisely. Given a base learner regression forest algorithm f and datastream $(x_i, y_i)_{i=1}^n$, the goal is to learn an interval-valued function $\Gamma(X_i, \alpha) : \mathbb{R}^d \times (0, 1) \mapsto I$, with the constraint

$$\mathbb{P}_{(X,y)\sim\pi}(y \notin \Gamma(X, \alpha)) \leq \alpha,$$

subject to constant excess model memory, prediction, and training time relative to the base learner's respective costs. We refer to α as the significance level, and $1 - \alpha$ as the confidence level.

In the next two subsections, we introduce two *batch* methods meant to augment regression forest algorithms with confidence intervals. In the section following, we discuss how the authors adapt these two batch methods to the online setting to achieve our aim.

1.2 Conformal prediction

Conformal prediction (CP) was first introduced in the batch setting by Papadopoulos et al. [2002]. The main idea behind conformal predictors in the batch setting is to estimate the confidence intervals from the empirical distribution of $|y - \hat{y}|$. Specifically, given batch data $\{(x_i, y_i)\}_{i=1}^N$, we set aside a subset C which we call the *calibration set*. After training our model f on the remaining data, we compute the set of *non-conformity scores*

$$S = \{|f(x_i) - y_i| : (x_i, y_i) \in C\}.$$

We then use S as an approximation of the true distribution of $|y - \hat{y}|$, and use S to compute the size of our confidence interval. Specifically, given α, we sort the conformity scores $s_1 \leq \cdots \leq s_n$, and choose our confidence interval by

$$\Gamma(x, \alpha) = [f(x) - s_{\lfloor (1-\alpha)n \rfloor}, f(x) + s_{\lfloor (1-\alpha)n \rfloor}].$$

Therefore, assuming the uniform distribution on S is a good approximation to the true distribution of $|y - f(x)|$,

$$\begin{aligned}
\mathbb{P}_\pi(y \notin \Gamma(x, \alpha)) &= \mathbb{P}_\pi(|y - f(x)| > s_{\lfloor (1-\alpha)n \rfloor}) \\
&\approx \mathbb{P}_{s \sim \text{Uniform}(S)}(s > s_{\lfloor (1-\alpha)n \rfloor}) \\
&= 1 - \frac{1}{n} \lfloor (1-\alpha)n \rfloor \\
&\leq \alpha,
\end{aligned}$$

as desired.

The main issue with moving CP online is it requires a growing subset of the datastream for the calibration set. Most naively this would necessitate $O(\lambda n)$ model memory and interval prediction compute, where λ is the probability that a training sample is used in the calibration set and n is the number of datapoints so-far seen. A second issue with conformal prediction is the intervals $\Gamma(x, \alpha)$

computed use no information about x or $\hat{f}(x)$: for a given calibration set the interval widths are solely a function of α. The first issue is resolved by the authors' online adaptation OnlineCP, and the second shortcoming is addressed by Quantile Regression Forests. Vovk et al. [2005] contains an overview of conformal prediction for the interested reader.

1.3 Quantile regression forests

Quantile Regression Forests (QRF), first introduced by Meinshausen [2006] is another meta-algorithm that takes as its base learner a regression forest, and generates an approximate conditional distribution $\pi_{y|x}$ of the output, rather than just a best guess of the ouput $\hat{y}(x)$. The desired confidence interval $\Gamma(x,\alpha)$ is then computed from this conditional distribution.

QRF works by storing label values at each leaf, rather than just their mean in a standard regression forest. To understand QRF in the general case, first begin by imagining that our base regression forest F consists of a single tree T, whose leaves are populated with already-seen training labels $\{y_i\}_{i=1}^N$. Given an unseen data vector x which is routed by T to some leaf $\text{leaf}_T(x)$, which in turn stores labels $\{y_1 \dots y_{n_l}\}$, our best guess of the conditional distribution $\pi_{y|x}$ is just the uniform distribution over the labels:

$$\hat{\pi}_{y|x,T}(x) := \text{Uniform}(\{y_{n_1}, \dots y_{n_l}\}) = \frac{1}{n_l}\sum_{i=1}^{n_l} \delta_{y_{n_l}}.$$

If F consists of several trees, we just approximate $\pi_{y|x}$ by a mixture distribution over the individual tree distributions:

$$\hat{\pi}_{y|x,F}(x) := \frac{1}{|F|}\sum_{T\in F} \pi_{y|x,T}(x).$$

This is the distribution used by QRF to compute confidence intervals.

Once our estimate of the conditional distribution $\pi_{y|x}$ is complete, we compute $\Gamma(x,\alpha)$ via quantiles of the estimated distribution. Let F be the CDF of the distribution $\hat{\pi}_{y|x,F}$ computed above. Defining the β^{th} quantile as

$$Q_\beta(x) = \inf\{y : F(y) \geq \beta\}.$$

We then define

$$\Gamma(x,\alpha) := [Q_{\alpha/2}(x), Q_{1-\alpha/2}(x)]$$

So that, assuming $\hat{\pi}_{y|x,F}$ is a good approximation to $\pi_{y|x}$,

$$\begin{aligned}\mathbb{P}_{\pi}(y \notin \Gamma(x, \alpha)|X = x) &= \mathbb{P}_{\pi_{y|x}}(y \notin [Q_{\alpha/2}(x), Q_{1-\alpha/2}(x)]) \\ &\approx \mathbb{P}_{\hat{\pi}_{y|x,F}}(y \notin [Q_{\alpha/2}(x), Q_{1-\alpha/2}(x)] \\ &\leq \alpha\end{aligned}$$

as desired.

As with CP, the main issue with moving QRF online is the model memory cost. Since all training labels are stored, model memory cost is $O(n)$, with n the number of samples so-far-seen. Unlike CP, QRF uses conditional information from x to compute the size of the confidence intervals. But this is not a strict advantage; it is a tradeoff. While QRF interval prediction intuitively should perform better on datasets where confidence interval size varies with respect to x, CP should perform better on more uniform datasets, especially when using low values of α to be sensitive to outliers, since CP stores a much larger range of labels both in the range of their corresponding feature vectors and in sheer number.

1.4 Mondrian Forests

We later compare online variants of the above two algorithms with a Mondrian Forests [Lakshminarayanan et al., 2016] baseline. Mondrian Forests is the state-of-the-art online regression forest algorithm capable of generating confidence intervals at the time of the author's writing [Vasiloudis et al., 2019]. Mondrian Forests model each tree as a hierarchical model of Gaussians, with a single Gaussian at each node instead of each leaf of the forest. The predictions are then modeled as a normally-distrbuted random variable, which permits computation of confidence intervals via quantiles. Mondrian Forests require $O(\log(n))$ compute time to process the n^{th} point in a datastream, in contrast to the $O(1)$ methods developed later in the paper.

The goal of the authors is to show that the methods they develop, OnlineCP and OnlineQRF, outperform Mondrian Forests in error rate and perform well relatively with respect to concept drift, while offering better memory and runtime guarantees than Mondrian Forests.

2 Methods

The authors modify CP and QRF to make them suitable for the online setting with OnlineCP and OnlineQRF. OnlineCP has two variants, CPExact and CPApprox, which trade off against each other in accuracy and speed. In this section I also introduce a minor but powerful modification of my own to CPExact, which provides a drastic runtime speedup while maintaining almost all of its accuracy and adaptability advantages over OnlineCP.

In exchange for $O(1)$ model memory and interval prediction compute time, OnlineCP and OnlineQRF give up their batch counterparts' asymptotic consistency. Instead the best these algorithms can do is give probabilistic accuracy guarantees.

2.1 Method 1: OnlineCP

2.1.1 Main Modifications to Batch CP

The authors modify the batch CP algorithm in three major ways:

Poisson training: In the batch setting, one has a finite training set D, from which one samples with replacement λD times to train each different tree T in a regression forest. Given a datapoint $(x, y) \in D$, we note that the number of times n it appears in a given tree's training is distributed according to

$$n \sim \text{Binomial}(\lambda|D|, \frac{1}{|D|}) \approx \text{Poisson}(\lambda)$$

for D large. Notably, this distribution is independent of the size of D, which allows us to adapt the training to the online setting. Moreover, it allows us to make principled (random) decisions to sort incoming datapoints into the calibration set or to prequential evaluation.

With this in mind, we modify training as follows: for each new datapoint (x_i, y_i), and each tree T in our forest, sample $k \sim \text{Poisson}(\lambda)$. If $k = 0$, put (x_i, y_i) in the calibration set. If $k > 0$, use (x_i, y_i) to train T with weight n. This method is also known as online bagging and was developed by Oza and Russell [2005].

Finite calibration set: To keep model memory constant, we limit our calibration set C to be a finite size FIFO queue. When C is full, it ejects its oldest example prior to adding a new one.

Out-of-bag calibration: When computing the non-conformity scores $|y_i - \hat{y}(x_i)|$, we must ensure that we only use trees T that have not seen the example (x_i, y_i) - otherwise we will underestimate the absolute error. To do so, for each point (x_i, y_i) in the calibration set, we keep a set l_{oob} of those trees T which were not trained on (x_i, y_i), and generate calibration predictions from this restricted forest rather than the whole forest. In accordance with the authors' writing, we refer to l_{oob} as the "out-of-bag" set of learners, whose etymological origin is the "bagging" procedure used to train random forests. This idea was originally introduced as the out-of-bag conformal regression algorithm in [Johansson et al., 2014].

2.1.2 Algorithms

With the above modifications aside, OnlineCP training, prediction, and conformity score computation is the same as their batch CP counterparts. We now provide specifications for the OnlineCP training and prediction algorithms. Note with $|F|$ the forest size and $|C|$ the maximum calibration set size that training, interval prediction, and conformity-score updating respectively consume $O(|F|)$, $O(|F|)$ and $O(|F||C|)$ compute time in excess of our base learner, and that the model memory is $O(|F| + |C|)$ in excess of the base learner. With $|F|$ and $|C|$ fixed constants, these meet our $O(1)$ memory and runtime requirements.

Algorithm 1 OnlineCP Training($F, (x, y), \lambda$)

```
Require: $F$ the regression forest, $C$ the calibration examples, $(x, y)$ a training example, $l_{oob}$ a map
   from calibration examples to learners
   $is\_oob$ = False
   for $T \in F$ do
       if $k = \text{Poisson}(\lambda) > 0$ then
           Train $T$ with $(x, y)$, weighted by $k$
       else
           $is\_oob$ = True
           Add $T$ to $l_{oob}[(x, y)]$
   if is_oob = True then
       Enqueue $(x, y)$ in $C$                                ▷ C is a FIFO queue of max size $c_{\max}$
```

Algorithm 2 OnlineCP Interval Prediction (F, x, α)

```
Require: $F$ the regression forest, $C$ the calibration examples, $x$ a feature vector, $\alpha$ the confidence
   level
   for $T \in F$ do
       Compute prediction $f_T(x)$
   $\hat{y} = \frac{1}{|F|} \sum_{T \in F} f_T(x)$
   Compute sorted non-conformity scores $S$
   $\phi = S[\lfloor (1 - \alpha)|S| \rfloor]$                 ▷ Compute interval width from $S$-quantiles
   return $\Gamma(x, \alpha) = [\hat{y} - \phi, \hat{y} + \phi]$
```

Algorithm 3 OnlineCP Update Non-conformity scores (F)

```
Require: $F$ the regression forest, $C$ the calibration examples, $l_{oob}$ a mapping from calibration
  examples to out-of-bag learners
  for $(x, y) \in C$ do
      for $T \in l_{oob}[(x, y)]$ do
          $\hat{y}_T = f_T(x)$                                ▷ Compute out-of-bag prediction
      $\hat{y}(x) = \frac{1}{|l_{oob}[(x,y)]|} \sum_{T \in F} \hat{y}_T$
  $S = |\mathbf{y} - \hat{\mathbf{y}}|$
```

2.1.3 CPExact and CPApprox

A question left intentionally ambiguous so far: each time the forest is trained on a new datapoint, do we recompute the calibration scores

$$S = \{|f(x) - y|, (x, y) \in C\}?$$

If we don't, as f is trained, the calibration scores may grow to be a poor approximation to the true distribution of the model's absolute error. If we do, we incur an $O(|F||C|)$ training cost to process a single datapoint compared to an $O(|F|)$ one without updating the calibration scores. In our experiment $|C| = 1000$, which slows down our processing time drastically.

The authors resolve this ambiguity by implementing OnlineCP as two separate algorithms: CPExact recomputes the calibration scores on each forest retraining, while CPApprox does not. CPExact holds two principal advantages over CPApprox that compensate for its sluggishness:

1. *Early in streaming.* Early in the streaming process, the first several forest predictions $f(x_i)$ poorly approximate the true labels y_i, and come to much better approximate the labels if we update f after it has seen several examples.

2. *Nonstationary data.* If the underlying distribution generating the data changes over the course of streaming, then CPExact will adapt its confidence intervals much more quickly than CPApprox. In effect there is a lag of size $|C|$ between CPExact's response to the changing distribution and CPApprox's, which is significant when $|C|$ is large relative to the underlying distribution's pace of change.

Finally, I list a notable (although not incredibly clever) improvement to CPExact of my own. Note that the runtime difference between CPExact and CPApprox is solely due to recomputing the calibration scores for each new datapoint. What if instead of recomputing the scores at each new datapoint arrival, we recomputed the calibration scores at *every other* datapoint arrival? Doing so would speed up CPExact roughly by a factor of two, while only trivially reducing the lag-advantage of size $|C|$ that CPExact holds over CPApprox to $|C|-1$. More generally, if we update the calibration scores after k training instances, we get a roughly k-fold speedup of CPExact while reducing the lag advantage to $|C|-k$, retaining its desirable properties if k is small relative to $|C|$.

In the experiments we implement CPExact in this fashion, with an update threshold $k = 50$, based on the heuristic $k = 5\%|C|$. In the results below we demonstrate a 6-12.5x speedup in CPExact, measured by the difference between my and the authors' implementation in its runtime relative to the other algoriths. Pre-experiment I noted anecdotally on my own computer a ∼50x runtime speedup between CPExact with update thresholding and CPExact without, confirming the rough theoretical heuristic above. I will later update this paper with a short experiment and data on this point.

2.2 Method 2: OnlineQRF

2.2.1 Main modifications to batch QRF

Recall the main idea of batch QRF is to store training label values at the leaves of a regression forest rather than just their mean. The main obstacle we encounter with this method in the online setting is the $O(|D|)$ memory cost of this method, where $|D|$ is the side of the dataset.

Aside from implementing Poisson training as discussed with OnlineCP, the main idea behind OnlineQRF is instead of storing all the labels $\{y_1 \dots y_n\}$ routed to a leaf at that leaf, we store a *quantile sketch* of these labels $\hat{R}(y_1 \dots y_n)$, a data structure that approximates the empirical distribution of $\{y_1 \dots y_n\}$ with a much lower memory footprint.

More precisely, given a data stream $\{y_1 \dots y_n\}$, a quantile sketch with accuracy parameter ϵ is data structure $\hat{R}_\epsilon(y_1 \dots y_n)$ that accurately estimates the rank $R(y)$ of any $y \in \{y_1 \dots y_n\}$ up to a relative error of ϵ, i.e.

$$|\hat{R}(y) - R(y)| \leq n\epsilon \text{ for all } y \in \{y_1 \dots y_n\}.$$

Quantile sketches may also be randomized, in which case we introduce an additional probabilistic parameter δ and require that

$$\mathbb{P}(|\hat{R}(y) - R(y)| \geq n\epsilon) \leq \delta \text{ for all } y \in \{y_1 \dots y_n\}.$$

Quantile sketches are constructed with two properties in mind. First, the quantile sketch must have a low memory cost relative to the size of the data. Second, quantile sketches are constructed to be *mergeable*: if one's data Y is split into a disjoint union $Y_1 \cup Y_2$, and $\hat{R}_1$, $\hat{R}_2$ are two ϵ-accurate sketches (resp. ϵ, δ accurate sketches in the randomized case) relative to Y_1 and Y_2, there must exist an algorithm which merges $\hat{R}_1$ and $\hat{R}_2$ into a new sketch $\hat{R}$ which is ϵ-accurate (resp. ϵ, δ-accurate) on the whole of Y.

The authors use the KLL sketch, named after its three authors Karnin et al. [2016]. The KLL sketch is a randomized, mergeable quantile sketch algorithm that uses $O(\epsilon^{-1} \log\log(\delta^{-1}))$ memory, which is provably optimal. Although there are several novel ideas in the KLL paper, the main idea is to use a chain of *compactors*, which iteratively downsample the sorted stream $S = \{y_1 \dots y_n\}$ into dyadic sorted substreams $S_k = \{y_{j \cdot 2^k}\}_{j=1}^{n/2^k}$. For an excellent introduction to the ideas behind quantile sketches in general and the KLL sketch in particular, see [Anon., 2020].

2.2.2 Algorithm specification

Given that we store a quantile sketch at each leaf in the forest, training and interval prediction are simple. To train a confidence-interval-predicting regression forest F with a label (x, y), first train it as a regression forest, using Poisson training as described above. Then for each $T \in F$, add y to the sketch at $\text{leaf}_T(x)$, the leaf to which x is routed by T. To form an interval prediction $\Gamma(x, \alpha)$, aggregate the sketches H_T stored at $\text{leaf}_T(x)$ for each $T \in F$, then merge them into one sketch H_F. Predict $\Gamma(x, \alpha)$ via the quantiles given by H_F.

We give more precise specifications of these algorithms below. Note with $|F|$ the forest size and $K = O(\epsilon^{-1} \log\log(\delta^{-1}))$ the memory cost of a single sketch, training and interval prediction take $O(|F|)$ and $O(|F| + K)$ compute, and model memory is $O(|\text{leaves}| \cdot K)$ in excess of the base learner. With $|F|$ and K fixed constants, these meet our $O(1)$ memory and runtime requirements - provided we use an online base learner with $O(1)$ memory footprint.

Algorithm 4 OnlineQRF Training $(F, (x, y), \lambda)$

```
Require: $F$ the regression forest, $(x, y)$ a training example, $H$ the KLL quantile sketch
  for $T \in F$ do
      if $k = \text{Poisson}(\lambda) > 0$ then
          Train $T$ with $(x, y)$, weighted by $k$
          Add $y$ to $H_T$, the sketch at $\text{leaf}_T(x)$
```

Algorithm 5 OnlineQRF Interval Prediction (F, x, α)

```
Require: $F$ the regression forest, $x$ a feature vector, $\alpha$ the confidence level
  $\hat{H} = \emptyset$                                              ▷ Empty KLL sketch
  for $T \in F$ do
      $H_T$ = KLL sketch stored at $\text{leaf}_T(x)$
      $\hat{H} = \text{Merge}(\hat{H}, H_T)$
  $\Gamma(x, \alpha) = [Q_\alpha(\hat{H}), Q_{1-\alpha/2}(\hat{H})]$     ▷ Compute quantiles of final sketch return $\Gamma(x, \alpha)$
```

3 Experiments

We run two experiments to assess the performance of the three algorithms CPApprox, CPExact, and OnlineQRF against Mondrian Forests, an online regression forest algorithm which was state-of-the-art at the time of the authors' writing in 2016.

3.1 Experimental design

The first experiment assesses the algorithms' performance using the prequential evaluation framework on 20 small datasets ($|D| \sim 10^4$) using four metrics. The second experiment uses the same four metrics to assess the algorithms' performance using prequential evaluation on one large synthetic dataset ($|D| = 10^6$) with *concept drift*, where the underlying distribution changes over the course of streaming. On the small datasets, we perform each experiment ten times and report each metric's average performance on each dataset. For the concept drift data, we evaluate and plot each metric over the course of streaming with tumbling windows of size 10,000. We also report the average runtime of the algorithms on the small data.

3.2 Metrics

We measure model performance using four metrics: the mean error rate (MER), the relative interval size (RIS), the quantile loss (QL), and the utility. The latter two metrics are functions of the former. We explain these concisely in order:

$$\text{MER} = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{y_i \in \Gamma(x_i, \alpha)}$$

The MER measures how often our confidence intervals miss our labels. We attempt to minimize the MER and note that if our algorithms work properly, the MER is bounded above by α.

$$\text{RIS} = \frac{1}{n} \sum_{i=1}^{n} \frac{\text{length}(\Gamma(x_i, \alpha))}{\text{range}(\{y_1 \dots y_n\})}$$

Note that the MER can be "gamed" by making the confidence intervals very large, regardless of the value of α. Therefore we also track the RIS, a measure of how tight the confidence intervals are relative to the range of the data.

$$\text{QL} = \alpha \cdot \text{RIS} + \frac{1}{n} \sum_{i=1}^{n} \frac{d(y, \Gamma(y_i, \alpha))}{\text{range}(\{y_1 \dots y_n\})} \qquad\qquad d(y, [a, b]) = \inf_{x \in [a,b]} |y - x|$$

The quantile loss used by the authors is a modified version of the quantile loss used in quantile regression [Koenker, 2005]. It is a holistic and suitably scaled way to combine the MER, scaled by error size, with the RIS.

$$\text{Utility} = \begin{cases} 1 - \text{RIS} & \text{MER} \leq \alpha \\ (1 - \text{RIS}) \cdot \exp(-\gamma(\text{MER} - \alpha)) & \text{otherwise} \end{cases} \qquad\qquad \gamma = \frac{2\log(2)}{\alpha}$$

Finally, the utility is based on the time-utility functions used in real-time systems [Ravindran et al., 2005], in which one aims to maximize a utility function with a penalty for doing so later in time. Here, the utility is 1-RIS, which holisitically measures the joint tightness of intervals an the absolute error size in a scale-invariant fashion. The "time" variable is the MER, with cutoff the significance level. After this cutoff, we exponentially penalize the utility according to the time after the cutoff, with half-life at MER $= 1.5\alpha$.

3.3 Implementation

The authors' code is open source and is available on GitHub. The authors modified the open-source Python library scikit-garden to implement Mondrian Forest meta-learners, and modified the FIMT-DD base learner classes in the open-source Java library Massive Online Analysis to implement the meta-learners OnlineCP and OnlineQRF.

My goal with the project was to implement everything in a completely Python-native fashion. To do so, I implemented the meta-learners OnlineCP and OnlineQRF by modifying the classes associated with the base learner AdaptiveRandomForestRegressor in River, an open-source Python library for online machine learning. I then made minor modifications to the authors' scikit-garden fork to make the Mondrian Forests compatible with the River streaming API. The whole experiment is implemented in two Jupyter notebooks in an open-source GitHub repository. This repository includes as submodules my fork of River and fork of scikit-garden.

To be clear (since both STAT 572 instructors instructed I do so): OnlineCP and OnlineQRF are *not* in any way natively implemented in River. I modified one of the supplied base learners - an adaptive version of the Hoeffding Tree Forest - to implement these. The only outside work I used in my code was the authors' implementation of the Mondrian Forest baseline, and snippets of the authors' results-plotting code to match the style of the original paper.

3.4 Data

For the small-scale stationary data experiments, we use 20 regression datasets from the OpenML repository [Vanschoren et al., 2014]. The datasets are both synthetic and real-world from domains such as robotics and housing, vary in size from $|D| = 4177$ to $|D| = 40768$, and contain as few as 8 and as many as 251 features. The full datasets are in the repository here, and we refer the reader to Appendix A of the authors' paper [Vasiloudis et al., 2019] for detailed descriptions of each dataset.

The concept-drift data is generated by the Friedman #1 function from [Ikonomovska et al., 2011]. For each datapoint in the stream, the features $(x_1 \ldots x_{10})$ are sampled from the uniform distribution over [0,1]. Then, the corresponding labels are generated according to the formula

$$y = 10 \cdot \sin(\pi x_1 x_2) + 20 \cdot (x_3 - 0.5)^2 + 10x_4 + 5x_5 + U,$$

where $U \sim \text{Uniform}([0, 1])$ is uniform random noise. Note that only the first 5 features affect the label. We sample 1 million datapoints in this fashion.

To create our final concept-drift dataset, we apply abrupt, global concept drift to the stream above. In particular, after 250k samples from the stream, we permute the features according to

$$(x_1, x_2, x_3, x_4, x_5) \mapsto (x_4, x_5, x_2, x_1, x_3).$$

After 750k total samples, we undo the permutation and revert to the original dataset.

3.5 Parameters

We follow the authors' parameter choices closely. For all models we set the forest size $|F| = 10$. We use $|C| = 1000$ calibration examples for OnlineCP, and implement CPExact using the sped-up method described above with an update threshold of $k = 50 = 0.05 \cdot |C|$. In OnlineQRF we construct our sketches based on accuracy parameter $K = 200$ (which depends implicitly on ϵ, δ). This was the choice made by the paper authors, is recommended by [Karnin et al., 2016], and is the default value in the Apache Datasketches library [dat] we used in the code - doing so yields a normalized rank error of $\epsilon \approx 1.65\%$.

We used River's AdaptiveRandomForestRegressor as our base learner, which implements the adaptive random forest algorithm by Gomes et al. [2017], designed specifically for online streaming that may contain concept drift.

4 Results

4.1 Small-scale data

We begin by reviewing our algorithms' performance on the small-scale data, summarized in Figures 1,2, and 3.

Examining Figure 1, we note Mondrian Forests' much larger average mean error rate, exceeding the specified significance $\alpha = 0.1$ on 12 out of 20 datasets. In contrast, OnlineQRF, CPExact, and CPApprox maintain an MER below α on all but two, one, and zero datasets respectively. Additionally, Mondrian Forests tends to produce much tighter confidence intervals than the authors' online algorithms - in Figure 1 we see that in 4 datasets, all the authors' algorithms produce mean interval sizes that are larger than 50% of the entire data range. We also note that the (MER, RIS) pairs for CPExact are roughly those of CPApprox, shifted down and to the right with a much lower spread in MER. CPExact maintains the highest and most consistent MER while staying below the threshold α.

Figure 2 shows that all methods perform similarly in quantile loss. Taking the QL as a holistic, correctly-scaled metric that simultaneously penalizes interval size and absolute error, we interpret this finding as saying that all methods have similar overall performance on the small-scale data - despite some methods performing better on MER and others better on RIS. Figure 3 shows a drastic difference in utility between the authors' three methods and Mondrian Forests. This difference is due to Mondrian Forests' high mean error rate, causing exponential decay in utility despite lower

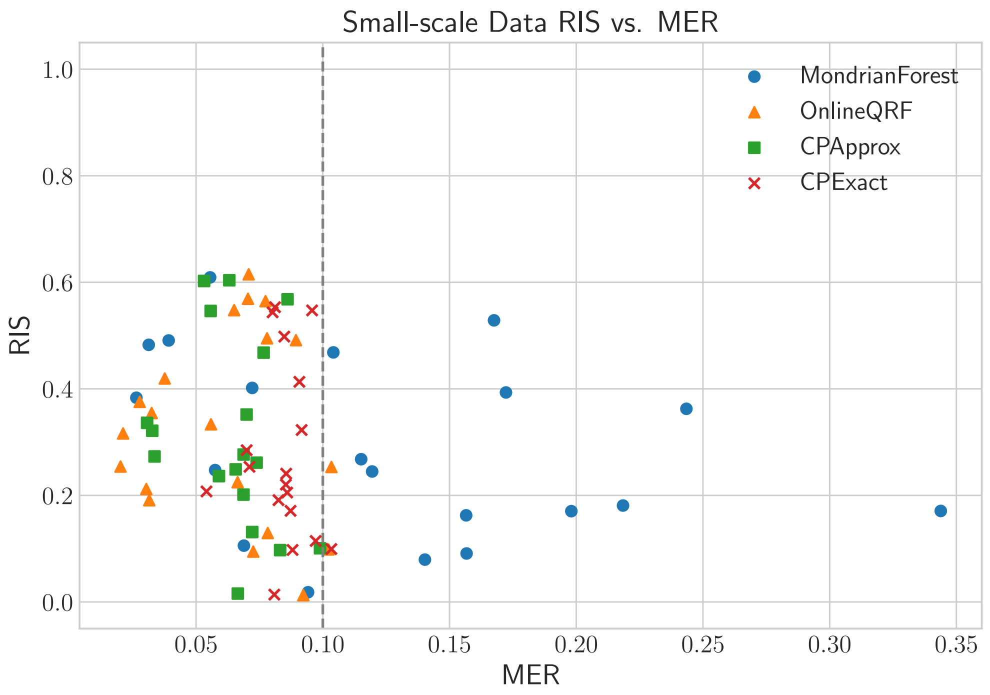

Figure 1: Each point corresponds to the average MIR and RIS of a method on one dataset, averaged over ten repeat experiments. We include a dashed line at significance level $\alpha = 0.1$.

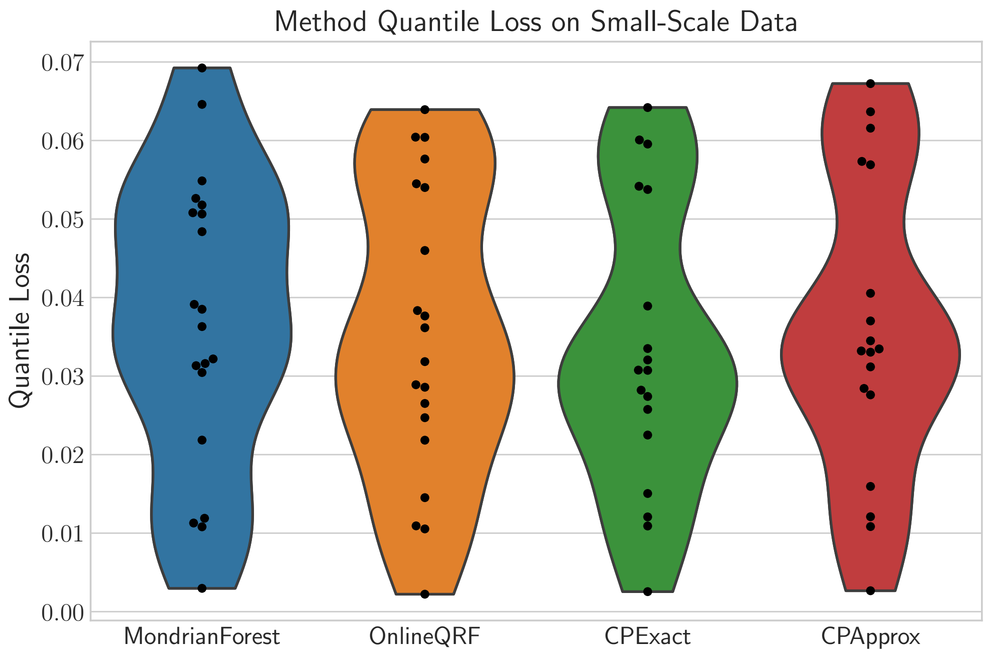

Figure 2: Quantile loss on small-scale data. Each point corresponds to the average quantile loss on one dataset, averaged over ten repeat experiments.

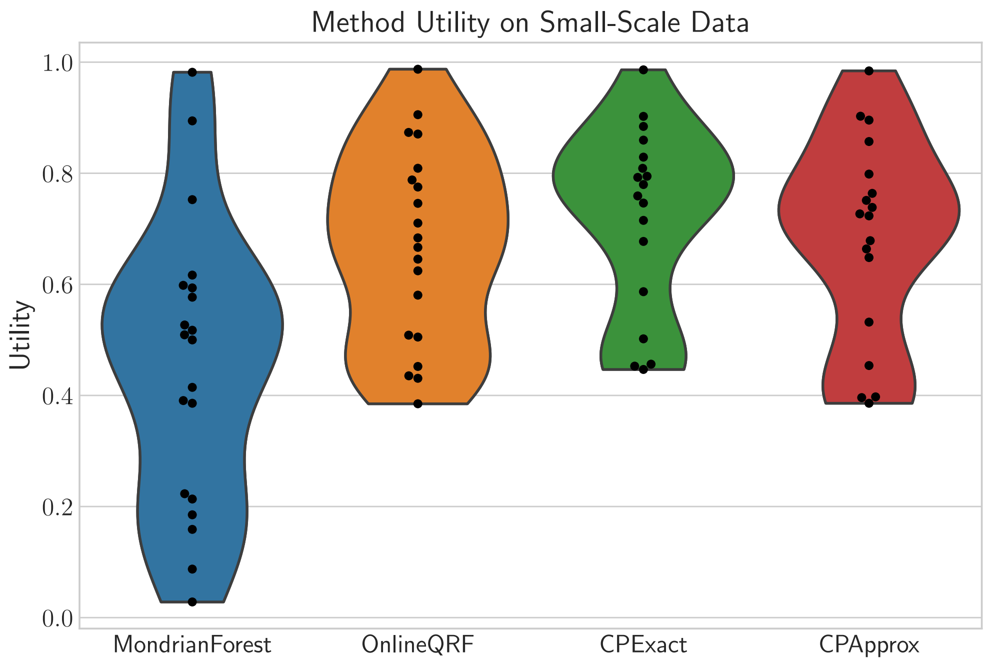

Figure 3: Utility on small-scale data. Each point corresponds to the average utility on one dataset, averaged over ten repeat experiments.

RIS than the authors' methods.

These findings can be summed up as follows: Mondrian Forests produce tight confidence intervals that frequently violate the requested significance, while the authors' algorithms produce typically valid confidence intervals that spread over a wider range of the data. Balancing these two error modes against each other with the quantile loss, all algorithms perform similarly. One explanation for this phenomenon is that the Mondrian Forest algorithm models the outputs $\hat{y}(x)$ as a Gaussian random variable (whose mean and standard deviation depends on the path taken from x to the various leaves $\text{leaf}_T(x)$). As a result, the distribution from which the confidence intervals are estimated will be thin-tailed. Therefore the confidence intervals will be tight but often invalid, especially with respect to fat-tailed (or just super-Gaussian) data.

4.2 Concept-drift data

Here we summarize the results of the concept drift experiment in Figures 4 and 5.

Immediately after abrupt concept drift is produced, all methods' performances suffer, but along different metrics. All methods exhibit an increase in MER, but OnlineQRF and Mondrian Forests suffer much more drastically than do the two variants of OnlineCP, spiking by 17% on an absolute

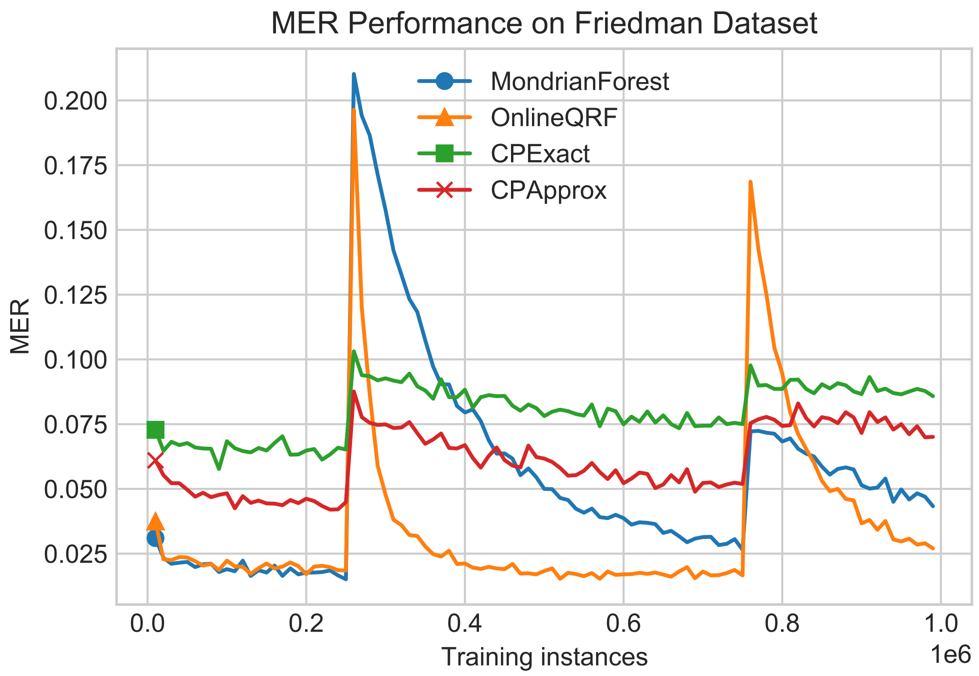

Figure 4: MER is computed on tumbling windows of size 10^6 while streaming the Friedman concept drift dataset.

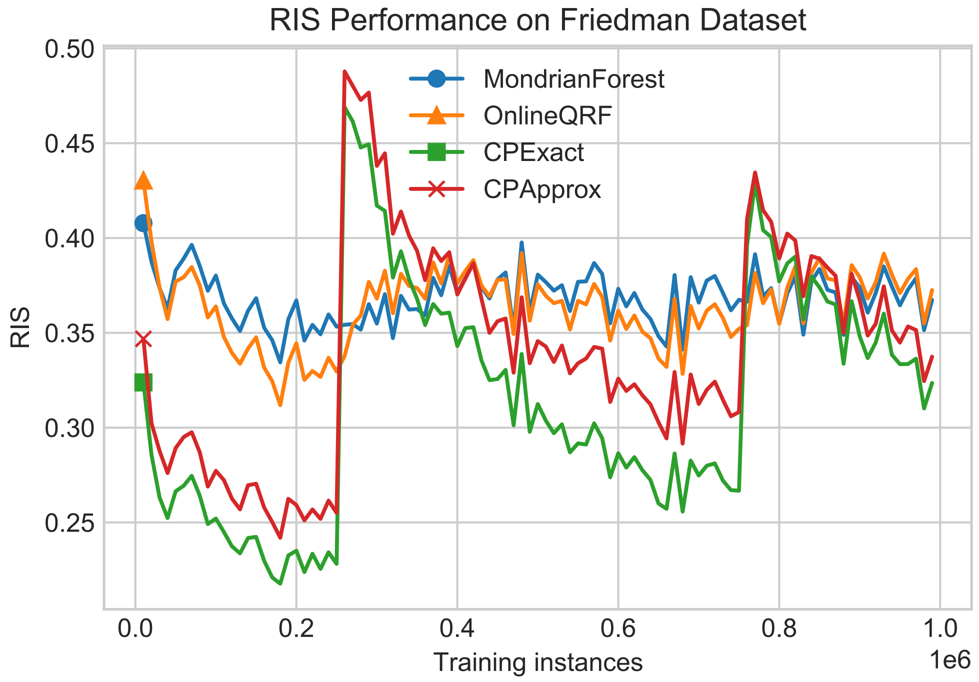

Figure 5: RIS is computed on tumbling windows of size 10^6 while streaming the Friedman concept drift dataset.

basis rather than 4-5%. On the other hand, both variants of OnlineCP nearly double their RIS at the time of concept drift, while Mondrian Forests and OnlineQRF maintain more stable RIS between 0.32 and 0.4 over the course of the experiment.

Both variants of OnlineCP maintain validity over the course of streaming, even immediately after abrupt concept drift. OnlineQRF and Mondrian Forests on the other hand generate invalid intervals after concept drift with a peak MER of 0.2, and differ in recovery time: OnlineQRF recovers to a below-target MER after 25,000 training samples, while Mondrian Forests takes over 100,000 training samples to do so.

In my implementation of the concept-drift experiment CPApprox and CPExact perform similarly, while they perform very differently in the authors' experiment. In the authors' paper, CPApprox exhibits a drastic MER spike ($> 40\%$ at its peak) while CPExact remains valid. Also, in the authors' paper CPExact's RIS spikes to over 1 (i.e., with confidence intervals larger than the range of the entire seen dataset) for over 200,000 training samples after concept-drift, while my imlementation exhibits nothing of the sort. This difference in results can't be due to my sped-up implementation of CPExact - if it was, I'd see worse performance and greater concept-drift lag in my implementation than that of the authors, but in fact I see the reverse. The origin of this difference in results is unclear, but my best guess is that it's due to a difference in base learners: I used River's AdaptiveRandomForestRegressor which implements the ARF algorithm in [Gomes et al., 2017], while the authors use the FIMT-DD algorithm as their base learner.

4.3 Runtime

We briefly list runtime results for the experiments and compare them with those of the authors in Table 1.

Dataset	MondrianForest	CPExact	CPApprox	OnlineQRF
Small-scale	185	74	50	49
Small-scale (authors')	42	102	5.7	5.4
Friedman	9862	9456	6576	6425
Friedman (authors')	2380	2011	213	114

Table 1: Average runtimes (seconds) for all methods in my and the authors' implementation.

In my implentation, all methods produce comparable runtime on the large dataset, while in the authors' paper CPApprox and OnlineQRF run approximately 10-20 times faster than Mondrian Forests and CPExact. This mostly has to do with shortcomings in my implementation of the large-scale experiment - I grew a large array that recorded the results of each prequential evaluation, which drastically slowed down each method by the same constant amount. As a result, the bulk of the large-scale experiments' runtime is spent on the same memory operations, causing comparable runtimes across methods.

The runtimes in my implementation also are not a fair comparison of the methods' performance due to the use of parallelism. In particular, the MondrianForest algorithms are implemented with parallelism across the $|F| = 10$ forest members, while I did not implement OnlineCP or OnlineQRF with parallelism. Therefore my results portray Mondrian Forests as faster, on a relative basis, than they really are. The authors implement all algorithms with parallelism, making their results a more accurate assessment of the algorithms' relative speeds.

Finally, we note the drastic runtime speedup due to my implementation of CPExact with an update threshold of $k = 50$, corresponding to 5% of the calibration set size. As we saw in the concept drift and small-scale data results, my implementation of CPExact performed similarly if not better to that of the authors. Thus while maintaining accuracy and interval tightness we obtain a 6-12.5x runtime speedup relative to its runtime against other algorithms in the authors' implementation.

5 Conclusion

The three online algorithms developed by the authors Vasiloudis et al. [2019] substantially outperforms Mondrian Forests in error rate, consistently generating valid confidence intervals. Mondrian Forests produce tighter confidence intervals than the authors' algorithms, and when these two error modes are balanced against each other via quantile loss, all algorithms perform comparably. From a modeling perspective, we prefer conservative algorithms which perform at or below the requested error rate as preferable to algorithms that err much more frequently than requested, even if the latter algorithms exhibit higher precision. In this sense OnlineCP and OnlineQRF perform substantially better than the state-of-the-art at the time of the authors' writing.

OnlineCP and OnlineQRF maintain valid confidence intervals even in the midst of abrupt concept drift. Moreover, these algorithms recover faster from concept drift than Mondrian Forests.

With respect to complexity, OnlineCP and OnlineQRF maintain theoretical $O(1)$ model memory and runtime guarantees to process the n^{th} sample in a datastream, while Mondrian Forests requires $O(\log(n))$ model memory and runtime to do so. The actual runtime of all algorithms on small and large datasets are comparable, but this is mostly due to the shortcomings of my own implementation. We also find that my modification of CPExact substantially speeds up runtime without sacrificing accuracy or adaptability in the face of concept drift.

These results largely replicate the results of the authors. We differ in actualized relative runtimes of the four algorithms, and CPExact and CPApprox's relative performance on concept-drift data. In the latter case my implementations performed similarly to each other and generally better than the authors', while the authors' implementations performed very differently from each other in the face of concept drift.

References

Apache datasketches. URL https://datasketches.apache.org/.

Anon. A review of recent results in streaming quantiles, 2020. URL http://courses.csail.mit.edu/6.854/20/sample-projects/B/streaming-quantiles.pdf.

João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. Issues in evaluation of stream learning algorithms. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, page 329–338, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605584959. doi: 10.1145/1557019.1557060. URL https://doi.org/10.1145/1557019.1557060.

Heitor M Gomes, Albert Bifet, Jesse Read, Jean Paul Barddal, Fabrício Enembreck, Bernhard Pfharinger, Geoff Holmes, and Talel Abdessalem. Adaptive random forests for evolving data stream classification. *Machine Learning*, 106(9):1469–1495, 2017.

Elena Ikonomovska, Joao Gama, and Sašo Džeroski. Learning model trees from evolving data streams. *Data mining and knowledge discovery*, 23(1):128–168, 2011.

Ulf Johansson, Henrik Boström, Tuve Löfström, and Henrik Linusson. Regression conformal prediction with random forests. *Machine learning*, 97(1):155–176, 2014.

Zohar Karnin, Kevin Lang, and Edo Liberty. Optimal quantile approximation in streams. In *2016 ieee 57th annual symposium on foundations of computer science (focs)*, pages 71–78. IEEE, 2016.

Roger Koenker. *Quantile Regression*. Number 9780521608275 in Cambridge Books. Cambridge University Press, 2005. URL https://ideas.repec.org/b/cup/cbooks/9780521608275.html.

Balaji Lakshminarayanan, Daniel M Roy, and Yee Whye Teh. Mondrian forests for large-scale regression when uncertainty matters. In *Artificial Intelligence and Statistics*, pages 1478–1487. PMLR, 2016.

Nicolai Meinshausen. Quantile regression forests. *Journal of Machine Learning Research*, 7(35): 983–999, 2006. URL http://jmlr.org/papers/v7/meinshausen06a.html.

Nikunj C. Oza and Stuart J. Russell. Online bagging and boosting. *2005 IEEE International Conference on Systems, Man and Cybernetics*, 3:2340–2345 Vol. 3, 2005.

Harris Papadopoulos, Kostas Proedrou, Volodya Vovk, and Alex Gammerman. Inductive confidence machines for regression. In Tapio Elomaa, Heikki Mannila, and Hannu Toivonen, editors, *Machine Learning: ECML 2002*, pages 345–356, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. ISBN 978-3-540-36755-0.

Binoy Ravindran, E Douglas Jensen, and Peng Li. On recent advances in time/utility function real-time scheduling and resource management. In *Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)*, pages 55–60. IEEE, 2005.

Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. OpenML. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, jun 2014. doi: 10.1145/2641190.2641198. URL https://doi.org/10.1145%2F2641190.2641198.

Theodore Vasiloudis, Gianmarco De Francisci Morales, and Henrik Boström. Quantifying uncertainty in online regression forests. *Journal of Machine Learning Research*, 20(155):1–35, 2019. URL http://jmlr.org/papers/v20/19-006.html.

Vladimir Vovk, Alex Gammerman, and Glenn Shafer. *Algorithmic Learning in a Random World*. 01 2005. doi: 10.1007/b106715.