POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

A Complete Suite for Conformal Prediction of Simple and Complex Data in R, with some theoretical extensions

Tesi di Laurea Magistrale in
Mathematical Engineering - Ingegneria Matematica

Paolo Vergottini, 946834

Advisor:
Prof. Simone Vantini

Co-advisors:
Dott. Jacopo Diquigiovanni
Dott. Matteo Fontana
Prof. Aldo Solari

Academic year:
2020-2021

Abstract: Conformal Prediction methods produce distribution-free prediction sets in regression, requiring only *i.i.d.* regression data. While R packages implementing such methods for the univariate response framework have been developed (for instance **conformalInference**), this is not the case with multivariate and functional responses. **conformalInference.multi** and **conformalInference.fd** address this lacuna, by providing implementations of full conformal, split conformal, and novel extensions of jackknife+ and multi split conformal to deal with multivariate and functional data. The extreme flexibility of conformal prediction, which does not require any specific regression model, enables users to pass in any regression function as input while using basic regression models as reference. Additionally, native plot functions are embedded to visualize prediction regions. Lastly, our contributions to the package **conformalInference**, including Conformalized Quantile Regression, are described in the appendix.

Key-words: Conformal Prediction, Multivariate Functional data, Prediction set, Multi Split Conformal, Jackknife+, Conformalized Quantile Regression

1. Introduction

Given a regression method, which outputs the predicted value $\hat{y}$, how do I define a prediction set offering at least marginal coverage level $1 - \alpha$? Recently, powerful methods that eliminate the need for specific distributions of the data have been developed. Predictive tools of this type belong to the family of nonparametric prediction methods, of which conformal prediction methods are the most prominent members. A broad review of the theory of conformal prediction can be found in [29]. Due to their wide-applicability and flexibility these methods are increasingly used in the practice.
Python is the most common framework for implementing these methods. For example, the **nonconformist** and **libconform** libraries handle both classification and prediction for regression with univariate responses (further information can be found in [17] and [10] respectively). For R users there is only one solid package available: **conformalInference**[1], which deals with the univariate case. Details may be found in [24]. Still, there is no R package that is able to deal with multivariate or even functional response cases thoroughly and comprehensively. This is where our packages **conformalInference.multi** and **conformalInference.fd**[2] come into play. They are

[1] The package is not yet available on CRAN and is currently hosted on GitHub. The author is Prof. Ryan Tibshirani of the Statistics and Machine Learning Department of Carnegie Mellon University.
[2] Both packages are available on CRAN.

inspired by **conformalInference**[3] and share with it the same philosophy with regards to user interface (this will be discussed in subsection 3.1) .

In the following discussion we will consider a set of values $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n) \overset{i.i.d.}{\sim}$[4], and define $z_i = (x_i, y_i)\ \forall i$. We will discuss three responses:

1. Univariate response: $y \in R$
2. Multivariate response: $y \in R^q$
3. Multivariate functional response: $y \in \prod_{j=1}^{q} L^{\infty}(\tau_j)$, where τ_j is a closed and bounded subset of $R^{d_j}, d_j \in N_{>0}$. In particular y_i is a function $i = 1, ..., q$.

To simplify notation we will generalize $y \in \mathcal{Y}$, keeping in mind all the due differences among the various contexts. From now on, let us consider a regression algorithm $\mathcal{G}$, then $\hat{\mu} := \mathcal{G}(z_1, ..., z_n)$ is a regression model trained on the dataset $\{z_1, ..., z_n\}$. As a result, given a new input value x_{n+1}, then $\hat{\mu}(x_{n+1}) = \mathcal{G}(z_1, ..., z_n)(x_{n+1})$ is the predicted response.

2. Conformal prediction theory

Initially, we will review the key aspects of Conformal Prediction theory for regression in the univariate response case, and then we will transition to multivariate and functional response frameworks.

2.1. Non-conformity score

Non-conformity score is the cornerstone of the Conformal Prediction theory. Indeed in a specific regression framework, given a new value z_{n+1}, we can measure how unusual it is compared to all the other observations $\{z_1, ..., z_n\}$ through the medium of a non-conformity measure, a real-valued function $\mathcal{A}$, returning $\mathcal{A}(\{z_1, ..., z_n\}, z_{n+1})$ and assigning greater value to the most unusual points. In the regression framework the test value is $z_{n+1} = (x_{n+1}, y)$, where y is the candidate response at the input point x_{n+1}.
Let's consider the different scenarios:

- In the univariate case the classical non-conformity measure is the absolute value of residuals:

$$\mathcal{A}(\{z_1, ..., z_n\}, z_{n+1}) = |y - \hat{\mu}(x_{n+1})|$$

- In the multivariate case we have more options. Specifically we will consider either the Euclidean distance (l2 norm) or the Mahalanobis distance between the candidate response y and the fitted value $\hat{\mu}(x_{n+1})$. Additionally, we may also compute the maximum of the residual for the test value x_{n+1}, as $A(\{z_1, ..., z_n\}, z_{n+1}) = max(y - \hat{\mu}(x_{n+1}))$.
- In the functional case we decided to adopt the non-conformity score defined in [9]:

$$\mathcal{A}(\{z_1, ..., z_n\}\}, z_{n+1}) = \sup_{j \in \{1,...,q\}} \left(\operatorname*{ess\,sup}_{t \in \mathcal{T}_j} \left| \frac{y^j(t) - [\hat{\mu}^j(x_{n+1})](t)}{s^j(t)} \right| \right) \tag{1}$$

where q is the number of components of the multivariate functional response, $\mathcal{T}_j$ is the space over which the j$^{\text{th}}$ functional response takes value, $\hat{\mu}^j$ is the j$^{\text{th}}$ component of the regression estimator, and $s^j(t)$ is a modulation function for dimension j built with the observations in $\{z_1, ..., z_n\}$. This non-conformity score descends from the supremum and incorporates a scaling factor along each component of the response.
In all the presented non-conformity scores we can exploit **modulation functions**, as in (1), whose role is to scale the non-conformity score by taking into account the variability along all directions, in such a way that prediction regions may adapt their width according to the local variability of data. If we restrict to a simple multivariate response, we can also consider modulation factors as $s^1_{\mathcal{I}_1}, ..., s^p_{\mathcal{I}_1}$. Following the line of reasoning in [9], we considered three alternatives: *identity*, where no scaling is applied, *st-dev*, where we scale with the standard deviation every component of the response, and *alpha-max*, where we ignore the most extreme observations and modulates data on the basis of the remaining observations. *alpha-max* is meant to be used only in a split conformal framework (described in subsection 2.3), where the complete dataset is split in a training set ($\mathcal{I}_1$) and a validation set ($\mathcal{I}_2$). This modulation function is defined as:

$$s^j_{\mathcal{I}_1}(t) := \frac{max_{h \in \mathcal{H}_1} |y_{h,j}(t) - [\hat{\mu}^j_{\mathcal{I}_1}(x_{h,j})](t)|}{\sum_{j=1}^{p} \int_{\tau_j} max_{h \in \mathcal{H}_1} |y_{h,j}(t) - [\hat{\mu}^j_{\mathcal{I}_1}(x_{h,j})](t)| dt} \tag{2}$$

[3] Our contributions to this package are detailed in the Appendix Univariate Response.
[4] The i.i.d. condition may be replaced by the weaker assumption of exchangeability.

where $\mathcal{H}_1 := \{h \in \mathcal{I}_1 : sup_{j\in\{1,...,q\}}(sup_{t\in\tau_j}|y_{h,j}(t) - [\hat{\mu}^j_{\mathcal{I}_1}(x_{h,j})](t)|) \leq \gamma\}$ and γ is the $\lceil(|\mathcal{I}_1|+1)(1-\alpha)\rceil$ smallest value in the set $\{sup_{j\in\{1,...,q\}}(sup_{t\in\tau_j}|y_{h,j}(t) - [\hat{\mu}^j_{\mathcal{I}_1}(x_{h,j})](t)|) : h \in \mathcal{I}_1\}$.

2.2. Full conformal

The first prediction method we will discuss is **Full Conformal**. Essentially, for a given test observation x_{n+1}, the non-conformity score ranks how well a candidate point $z_{n+1} = (x_{n+1}, y)$ matches with all the rest of the data, selecting y from a grid of candidates. While theoretically this approach can be extended to multivariate response contexts (of dimension q), by considering a q-dimensional grid, in practice it scales poorly, having to test $|grid\ points|^q$ candidates. Thus, the approach can only be applied when q is fairly small. On top of that, a potential extension to the functional case suffers from a critical problem: one should consider as candidates all possible functions in $\prod_{j=1}^{q} L^\infty(\tau_j)$, where τ_j is a closed and bounded subset of $R^{d_j}, d_j \in N_{>0}$.
To compute all the non-conformity scores we need to construct an augmented regression estimator $\hat{\mu}_y := \mathcal{G}(\{z_1, ..., z_n, z_{n+1}\})$, trained over the augmented data set $\{z_1, ..., z_n, z_{n+1}\}$, and choose the non-conformity measure $\mathcal{A}$. With univariate data, we will consider the absolute value of the residuals, while with multivariate data we can take the l_2 norm as an example. We define the residual's scores as:

$$R_i := \mathcal{A}(\{z_1, ..., z_{i-1}, z_{i+1}, ..., z_n, , z_{n+1}\}, z_i) \quad R_{n+1} := \mathcal{A}(\{z_1, ..., z_n\}, z_{n+1}) \tag{3}$$

Our next step is to rank R_{n+1} with respect to all the other residual scores $R_1, ..., R_n$ and calculate the fraction of values with higher non-conformity scores than the candidate point as

$$\delta_y = \frac{|j \in \{1, ..., n, n+1\} \ : R_j \geq R_{n+1}|}{n+1} \tag{4}$$

Lastly, we identify the full conformal prediction set as $C_{full}(x_{n+1}) = \{y \in \mathcal{Y} \ : \delta_y > \alpha\}$. As shown in [16], the quantity $1 - \delta_y$ can serve as a valid p-value for testing the null hypothesis that $H_0 : Y_{n+1} = y$, and the following theorem holds:

$$P(y_{n+1} \in C_{full}(x_{n+1})) \geq 1 - \alpha \tag{5}$$

Therefore, the method guarantees a coverage level $1 - \alpha$. The pseudo-code for the method can be found in Algorithm 1.

Algorithm 1: Full Conformal Prediction

Input: level $\alpha \in (0, 1)$, and a regression algorithm $\mathcal{G}$
Data: $z_i = (x_i, y_i)\ for\ i = 1, ..., n$, test point x_{n+1}, and a set of candidate responses $\mathcal{Y}^c = \{y_1, y_2, ...\}$
Output: Prediction intervals at point x_{n+1}
foreach $y \in \mathcal{Y}^c$ **do**

$\hat{\mu}_y = \mathcal{G}(\{z_1, ..., z_n, (x, y)\})$
Compute nonconformity scores:
$R_i = |y_i - \hat{\mu}_y(x_i)| \ \ and \ \ R_{n+1} = |y - \hat{\mu}_y(x)|$ (Univariate)
$R_i = ||y_i - \hat{\mu}_y(x_i)||_2 \ \ and \ \ R_{n+1} = ||y - \hat{\mu}_y(x_{n+1})||_2$ (Multivariate)

$\delta_i = \frac{1+\sum_{k=1}^{n} \mathbb{1}(R_i) \leq R_{n+1}}{n+1}$

end

return $C_{full}(x_{n+1}) = \{y \in \mathcal{Y}^c : \delta_y \geq \alpha\}$

2.3. Split conformal

A computational less demanding method called **Split Conformal** was introduced to overcome the complexity of full conformal. Essentially the observations $y_1, ..., y_n$ are split into a training set $\mathcal{I}_1$ and a validation set $\mathcal{I}_2$, respectively of cardinality m and l. As a general rule, the split procedure is aleatory, though one could also pick the observations to be used in the training set. The first set is used to train, just once, the regression model as $\hat{\mu}_{\mathcal{I}_1} := \mathcal{G}(\{z_k : k \in \mathcal{I}_1\})$, while the second set is used to compute, for every $i \in \mathcal{I}_2$, the distance of each y_i from the fitted value $\hat{\mu}_{\mathcal{I}_1}(x_i)$ through a non-conformity measure. Indeed, consider

$$R_i := \mathcal{A}(\{z_k : k \in \mathcal{I}_1\}, z_i) \quad i \in \mathcal{I}_2 \qquad R_{n+1} := \mathcal{A}(\{z_k : k \in \mathcal{I}_1\}, z_{n+1}) \tag{6}$$

and use the new definitions of residuals' scores in (6) to obtain δ_i as in (4).
Hence, $C_{split}(x_{n+1}) = \{y \in \mathcal{Y} \ : \delta_y > \alpha\}$. As shown by [27], the following inequalities state the theoretical coverage of the split conformal method:

$$1 - \alpha \leq P(y_{n+1} \in C_{split}(x_{n+1})) < 1 - \alpha + \frac{1}{l+1} \tag{7}$$

Thus, split conformal prediction sets are valid, but there is no guarantee of exactness, i.e. having precise coverage of $1 - \alpha$. A slight modification can provide exactness: if we introduce a randomization element $\tau_{n+1} \sim \mathcal{U}_{[0,1]}$, we can express the smoothed split conformal predictions set $C_{split,\tau_{n+1}}(x_{n+1}) := \{y \in \mathcal{Y} : \delta_{y,\tau_{n+1}} > \alpha\}$, where

$$\delta_{y,\tau_{n+1}} = \frac{|\{j \in \mathcal{I}_2 : R_j > R_{n+1}\}| + \tau_{n+1}|\{j \in \mathcal{I}_2 \cup \{n+1\} : R_j = R_{n+1}\}|}{l+1} \tag{8}$$

The proof showing that the smoothed version leads to exact prediction sets for any value of α and l is in [8].
According to geometrical considerations[5] about split conformal, one might define $k := \lceil (1-\alpha)(l+1) \rceil$ and select the k^{th} smallest non-conformity score in $\{R_i \ : i \in \mathcal{I}_2\}$ as d. Then the alternative definition of the split conformal prediction set is

$$C_{split}(x_{n+1}) = [\hat{\mu}_{\mathcal{I}_1}(x_{n+1}) - d, \hat{\mu}_{\mathcal{I}_1}(x_{n+1}) + d] \tag{9}$$

Likewise, for the smoothed version one may define $k_{\tau_{n+1}} = \lceil l + \tau_{n+1} - (l+1)\alpha \rceil$ and select the $k^{th}_{\tau_{n+1}}$ smallest non-conformity score in $\{R_i \ : i \in \mathcal{I}_2\}$ as $d_{\tau_{n+1}}$. Thus, another way of defining the split conformal prediction set would be:

$$C_{split,\tau_{n+1}}(x_{n+1}) = [\hat{\mu}_{\mathcal{I}_1}(x_{n+1}) - d_{\tau_{n+1}}, \hat{\mu}_{\mathcal{I}_1}(x_{n+1}) + d_{\tau_{n+1}}] \tag{10}$$

Observe that if $\tau_{n+1} = 1$, we revert back to the classical split conformal version. The pseudo-code is shown in Algorithm 2.
As long as a suitable non-conformity measure is employed, the split conformal method can be applied successfully in the multivariate response and multivariate functional response frameworks as well.

Algorithm 2: Split Conformal Prediction

Input: split proportion ρ, level $\alpha \in (0,1)$, and a regression algorithm $\mathcal{G}$
Data: $z_i = (x_i, y_i)$ for $i = 1, ..., n$, and a test point x_{n+1}
Output: Prediction sets at each point in $\mathcal{X}$,

Randomly split $\{1, ..., n\}$ into $\mathcal{I}_1$ and $\mathcal{I}_2$ so that $|\mathcal{I}_1| = \rho n$ and $|\mathcal{I}_2| = \rho(1-n) =: l$
Build the regression function $\hat{\mu}_{\mathcal{I}_1} = \mathcal{G}(\{z_k \ : \ k \in \mathcal{I}_1\})$
Compute nonconformity scores for $i \in \mathcal{I}_2$:
$R_i = |y_i - \hat{\mu}_{\mathcal{I}_1}(x_i)|$ (Univariate)
$R_i = ||y_i - \hat{\mu}_{\mathcal{I}_1}(x_i)||_2$ (Multivariate)
$R_i = \sup_{j \in \{1,...,q\}} \left(\sup_{t \in \mathcal{T}_|} \left| \frac{y_{i,j}(t) - [\hat{\mu}^j_{\mathcal{I}_1}(x_{i,j})](t)}{s^j_{\mathcal{I}_1}(t)} \right| \right)$ (Functional)
Now choose between smoothed or classical version:
$k = \lceil (l+1)(1-\alpha) \rceil$ (Classical)
Sample $\tau_{n+1} \sim \mathcal{U}_{[0,1]}$ and $k = \lceil l + \tau_{n+1} - (l+1)\alpha \rceil$ (Smoothed)
d = k$^{\text{th}}$ smallest value in $\{R_k \ : \ k \in \mathcal{I}_2\}$

return $C_{split}(x_{n+1}) = [\hat{\mu}_{\mathcal{I}_1}(x_{n+1}) - d, \hat{\mu}_{\mathcal{I}_1}(x_{n+1}) + d]$

---

[5]These considerations are explicit in `https://www.intechopen.com/chapters/5294`

2.4. Jackknife+

The **Jackknife+** method was introduced in [2] to tackle the major limitations of full and split conformal. Full conformal is computational quite demanding, while split conformal exploits only the observations in the training set to train the model (almost "wasting" the observations in the validation set) and is stochastic: different runs of the algorithm yield different prediction regions. Jackknife+ manages to overcome these drawbacks and it still relies on the assumption of *i.i.d.* data. As this is a slight variation of the well-known Jackknife method, we should first review the classic jackknife prediction interval. Indeed, assuming $(x_1, y_1), .., (x_n, y_n)$ $i.i.d$:

$$C_{jack} = [q_\alpha\{\hat{\mu}(x_{n+1}) - R_i^{LOO}\},\ q_{1-\alpha}\{\hat{\mu}(x_{n+1}) + R_i^{LOO}\}] \tag{11}$$

where $R_i = |y_i - \hat{\mu}_{-i}(x_i)|\ \forall i = 1, ..., n$ are the absolute leave-one-out (or LOO) residuals and $\hat{\mu}$ is a regression model trained over the whole dataset $\{z_1, ..., z_n\}$. The LOO residuals can be computed by fitting a complete regression model and n leave-one-out models, removing one observation at a time for training. It provides symmetric intervals, since they are all centered around $\hat{\mu}(x_{n+1})$, and it tackles the issue of overfitting by using the quantile of the absolute residuals from LOO. Although fitting $n + 1$ models seems extremely challenging, some models have shortcuts for obtaining LOO residuals (for example, the linear model can use the Sherman-Morrison updating scheme).
The analysis of the theoretical coverage of jackknife and jackknife+ is discussed in [2]. The authors constructed an extreme case where (11) led to empty coverage, while they proved jackknife+ provides a guarantee of $(1-2\alpha)\%$ coverage. Let us now present the jackknife+ interval formulation:

$$C_{jack+} = [q_\alpha\{\hat{\mu}_{-i}(x_{n+1}) - R_i^{LOO}\},\ q_{1-\alpha}\{\hat{\mu}_{-i}(x_{n+1}) + R_i^{LOO}\}] \tag{12}$$

Unlike (11), the predictions are not centered on $\hat{\mu}(x_{n+1})$ but instead are shifted by $\hat{\mu}_{-i}(x_{n+1})$. This is most noticeable when the regression model is highly sensitive to training data. As previously mentioned we can show that:

$$P(y_{n+1} \in C_{jack+}(x_{n+1})) \geq 1 - 2\alpha \tag{13}$$

Since jackknife+ requires a univariate quantile, it is not straightforward to translate that concept to a multivariate or functional model. A possible solution revolves around the concept of non-conformity measure. In particular, as a measure of non-conformity for both multivariate and functional responses, we chose:

$$\begin{aligned} \mathcal{A}_{max}(x, y) &= \sup_{j\in\{1,\dots,q\}} \left| \frac{y_j - [\hat{\mu}^j(x)]}{s^j} \right| \quad (multivariate) \\ &= \sup_{j\in\{1,\dots,q\}} \left(\operatorname*{ess\,sup}_{t\in\mathcal{T}_j} \left| \frac{y_j(t) - [\hat{\mu}^j(x_j)](t)}{s^j(t)} \right| \right) \quad (functional) \end{aligned} \tag{14}$$

For the sake of simplicity, we will refer to $\mathcal{A}_{max}(x, y)$, while keeping in mind the two different formulations. In order to replicate the concept of quantile, we ordered the data points according to their non-conformity scores. This allows us to select as a multivariate and functional quantile ($q_\alpha^{\mathcal{A}}$), the level set induced by the non-conformity measure. In particular considering $u_1, .., u_n \in \mathcal{U}$, we exclude all the data points with non-conformity score greater than $q_{1-\alpha}\{\mathcal{A}_{max}(u_1), ..., \mathcal{A}_{max}(u_n)\}$:

$$q_\alpha^{\mathcal{A}}(u_1, .., u_n) := \{u \in \mathcal{U} : \mathcal{A}_{max}(u) \leq q_{1-\alpha}\{\mathcal{A}_{max}(u_1), ..., \mathcal{A}_{max}(u_n)\}\} \tag{15}$$

Note $q_{1-\alpha}\{\mathcal{A}_{max}(u_1), ..., \mathcal{A}_{max}(u_n)\}$ refers to the classical quantile. Moreover, this non-conformity measure behaves in the exact opposite way to that of a depth measure: in addition to the rating from "common" to more unusual data, it also provides an centered-outward ranking, with high scores assigned to extreme data. In any case, the ranking induced by (15) is not comparable with the smallest-to-largest values ranking in the univariate case. Consequently, the notion of extended quantile should take into account this interpretation change. $q_\alpha^{\mathcal{A}}$ is not always a "small" value (in fact, the concept of being small is meaningless in this framework), but rather an extreme or less conformal value. The converse reasoning can be applied to $q_{1-\alpha}^{\mathcal{A}}$.
As a result, in the multivariate context we could simply extend (12) as :

$$C_{jack+} = \{y \in R^q : y \in [q_\alpha^{\mathcal{A}}(\{\hat{\mu}_{-i}(x_{n+1}) \pm R_i^{LOO} : i = 1, ..., n\})]\} \tag{16}$$

While in the functional context:

$$C_{jack+} = \{y \in \prod_{j=1}^{q} L^{\infty}(\tau_j) : y(t) \in [q_{\alpha}^{\mathcal{A}}(\{\hat{\mu}_{-i}(x_{n+1}) \pm R_i^{LOO} : i = 1, ..., n\})(t)] \ \ \forall t \in \prod_{j=1}^{q} \tau_j\} \tag{17}$$

As shown in Algorithm 3, after obtaining the level sets with $q_{\alpha}^{\mathcal{A}}$, we compute the axis-aligned minimum bounding box (or AABB), effectively projecting the prediction region over the axis. More details about AABB can be found in [6]. The method yields a prediction interval for each component of the response, simplifying the interpretation of the results.

Algorithm 3: Jackknife+ Prediction

Input: level $\alpha \in (0, 1)$, and a regression algorithm $\mathcal{G}$
Data: $z_i = (x_i, y_i)$ *for* $i = 1, ..., n$, and a test point x_{n+1}
Output: Prediction set at point x_{n+1}
for *i=1, ... , n* **do**
  Train regression algorithm $\hat{\mu}_{-i} := \mathcal{G}(\{z_1, ..., z_{i-1}, z_{i+1}, ..., z_n\})$
  Compute LOO residual $R_i^{LOO} = |y_i - \hat{\mu}_{-i}(x_i)|$
end

$R^{-} := \{\hat{\mu}_{-i}(x_{n+1}) - R_i^{LOO} \ \ i = 1, ..., n\}$
$R^{+} := \{\hat{\mu}_{-i}(x_{n+1}) + R_i^{LOO} \ \ i = 1, ..., n\}$
$R^{\pm} := \{\hat{\mu}_{-i}(x_{n+1}) \pm R_i^{LOO} \ \ i = 1, ..., n\}$

$C_{jack+}(x_{n+1}) = [q_{\alpha}(R^{-}), q_{1-\alpha}(R^{+})]$ (Univariate)

$C_{jack+}(x_{n+1}) = BoundingBox(q_{\alpha}^{\mathcal{A}}(R^{\pm}))$ (Multivariate - Functional)

return $C_{jack+}(x_{n+1})$

2.5. Multi split conformal

Among the drawbacks of split conformal is the inherent randomness in the splitting procedure, since each split produces a valid prediction set. To overcome this limitation, the **Multi Split Conformal** method was developed. As described in [23], the split conformal method is run multiple times (B) and then the prediction intervals are combined. To determine the minimum number of intervals within which a point must be contained to be included in our final prediction set, we use τ. Formally, given the split conformal intervals $C^{[1]}, ..., C^{[B]}$, we can define $\Pi^y = \frac{1}{B} \sum_{b=1}^{B} 1\{y \in C^{[b]}\} \ \forall y \in R$. Then the multi split conformal prediction interval is:

$$C_{msplit}(x_0) = \{y \in R \ : \ \Pi^y > \tau\} \tag{18}$$

It is important to note that the split conformal intervals $C^{[1]}, ..., C^{[B]}$ are obtained by setting the miscoverage probability to $\alpha(1 - \tau + \lambda/B)$, where λ is a smoothing parameter (a positive integer). As proven in [23], the multi split prediction interval has coverage at least $1 - \alpha$.
The extension to the multivariate and functional case may be inspired by the one discussed in the previous subsection. Indeed, one runs the split conformal methods multiple times and joins their lower bounds and upper bounds into a single set. The next step employs the notion of quantile defined in (15). By considering only points with low non-conformity scores, we can build a level set induced by the ranking provided by $\mathcal{A}_{max}$. This time around, we will choose the level of the quantile based on $2\tau B$, i.e. twice the chosen amount of areas or bands a point must be contained in to be part of the final prediction set. For the final prediction set, we consider the axis-aligned bounding box of the produced output, just as we did with the jackknife+ extension. An efficient way to compute the multi split prediction interval, taken from [12], is shown in Algorithm 4.

Algorithm 4: Multi Split Conformal Prediction

Input: split proportion vector *prop*, level $\alpha \in (0,1)$, and a regression algorithm $\mathcal{G}$, number of replications B, smoothing parameter λ, joining parameter τ

Output: $z_i = (x_i, y_i)$ *for* $i = 1, ..., n$, and a test point x_{n+1}

Data: Testing set x

for b = *1, ... ,* B **do**

  $(C^{[b]}, lo^{[b]}, up^{[b]})$=Split($\{z_1, ..., z_n\}$, x_{n+1}, $prop[b]$, $\alpha(1 - \tau + \lambda/B)$, $\mathcal{G}$)

end

if *univariate case* **then**

  $\Pi^y = \frac{1}{B}\sum_{b=1}^{B} 1\{y \in C^{[b]}\}\ \forall y \in R$

  $C_{msplit}(x_{n+1}) = \{y \in R\ :\ \Pi^y > \tau\}$

end

if *multivariate-functional case* **then**

  $L = \{lo^{[b]}, up^{[b]}\ b = 1, ..., B\}$

  $L_q = q^{\mathcal{A}}_{2\tau B}(L)$

  $C_{msplit}(x_{n+1}) = BoundingBox(L_q)$

end

return $C_{msplit}(x_{n+1})$

Function `Split`($\{z_1, ..., z_n\}$, x_0, ρ, α, $\mathcal{G}$)**:**

  Randomly split $\{1, ..., n\}$ into $\mathcal{I}_1$ and $\mathcal{I}_2$ so that $|\mathcal{I}_1| = \rho n$

  Build the regression function $\hat{\mu}_{\mathcal{I}_1} := \mathcal{G}(\{z_k\ :\ k \in \mathcal{I}1\})$

  $R_i = |y_i - \hat{\mu}_{\mathcal{I}_1}(x_i)|$

  $k = \lceil n(1-\rho)(1-\alpha) \rceil$

  d = k-th smallest value in $\{R_k\ :\ k \in \mathcal{I}_2\}$

  return $C^{split}_{n,1-\alpha}(x_{n+1}) = [\hat{\mu}_{\mathcal{I}_1}(x_{n+1}) - d, \hat{\mu}_{\mathcal{I}_1}(x_{n+1}) + d]$,
  $lo = \hat{\mu}_{\mathcal{I}_1}(x_{n+1}) - d$, $up = \hat{\mu}_{\mathcal{I}_1}(x_{n+1}) + d$

3. conformalInference.multi : structure and functionality

The **conformalInference.multi** package performs conformal prediction for regression when the response variable y is multivariate. Table 1 contains a list of all available functions. Additionally, there are a few helper functions implemented in the package: for instance, the `check` functions verify the type correctness of input variables (if not, a warning is displayed to the user and the code execution is interrupted), `interval.build` joins multiple regions into a single one and is vital to the implementation of the multi split method, and `computing_s_regression` provides modulation values for the residuals in the conformal prediction methods. One can mainly identify two types of methods: regression methods and prediction functions.

Function	Description
`conformal.multidim.full`	Computes Full Conformal prediction regions
`conformal.multidim.jackplus`	Computes Jackknife+ prediction regions
`conformal.multidim.split`	Computes Split Conformal prediction regions
`conformal.multidim.msplit`	Computes Multi Split Conformal prediction regions
`elastic.funs`	Build elastic net regression
`lasso.funs`	Build lasso regression
`lm_multi`	Build linear regression
`mean_multi`	Build regression functions with mean
`plot_multidim`	Plot the output of prediction methods
`ridge.funs`	Build elastic net regression

Table 1: List of the main functions in **conformalInference.multi**. Helpers functions are excluded.

3.1. Regression methods

The philosophy behind this package is based on one major assumption: the regression methods should not be included in the prediction functions themselves, as the user may require fitting a very specific regression model. The most common regression methods have already been introduced. Yet, the modular structure of the package allows the user to employ custom-coded regression functions, according to the specific regression task at hand. To keep the problem simple, we assumed each dimension of the response to be independent, so each component was fitted with a separate model. Although this may seem restricting, native models can be built for multivariate multiple regression. Three different methods are available: the *mean model*, where the response is modeled using the sample mean of the observations, the *linear model* and the *elastic net model*, which applies the **glmnet** package and also contains lasso and ridge regression subcases. Two elements will be returned by these functions: `train.fun` and `pred.fun`. The first, given as input the matrices x and y (avoid data frames, as depending packages may not accept them), provides the training function, while the second, taking as input the test values and the output of `train.fun`, returns the predictions at the test values x_0.

3.2. Prediction methods

All the prediction functions take as input a matrix `x` of dimension n x p, a matrix of responses `y` of dimension n x q, a test matrix `x0` with dimension n_0 x p and the level of the prediction $\alpha \in (0, 1)$. With the string `score` we can select one of three non-conformity scores: *l2, mahalanobis, max*. Moreover, the scores can be scaled either using the *mad functions* or with the parameter `s.type`. The *mad functions*, i.e. `mad.train.fun` and `mad.predict.fun`, are user defined functions (with the same structure as the regression methods) that scale the residuals, whereas the parameter `s.type` determines a modulation function between *identity*, no residual scaling, *st-dev*, dividing the residuals by the standard deviation along various dimensions and *alpha-max*, which produces modulations similar to those of *st-dev*, but ignores the more extreme observations for the local scaling. In addition, a logical `verbose` allows to print intermediate processes. The package **future.apply**, which leverages the package **future** to parallelise computations, was used to speed up computations whenever parallelisation is needed.

The first implemented method is full conformal. The algorithm requires a grid of candidate response for each test value in `x0`. The grid points in dimension k are `num.grid.pts.dim` points equally spaced in the interval $[-grid.factor * max|y(k)|, grid.factor * max|y(k)|]$, where $y(k)$ is the value of the response in the k$^{\text{th}}$ dimension. For this range of trial values (assuming $grid.factor \geq 1$) the cost in coverage is at most $\frac{1}{n+1}$, as shown in [5]. Additionally, in full conformal the *alpha-max* option for `s.type` is not included. The function returns the predicted values and a list of candidate responses at each element in x_0. Moreover, responses will be accepted only when bivariate, to minimize computational overhead.

```
conformal.multidim.full = function(x, y, x0, train.fun, predict.fun,alpha = 0.1,
                                   mad.train.fun = NULL,mad.predict.fun = NULL,
                                   score='l2', s.type = "st-dev",
                                   num.grid.pts.dim=100, grid.factor=1.25,
                                   verbose=FALSE)
```

Alternatively, the split conformal method divides the dataset into a training set and a validation set, and this division is controlled by the argument `split`, a vector of indices for the training set, or by the argument `seed`, which initializes a pseudo random number generator for splitting. A randomized version of the algorithm can be selected using `randomized`, and the sampled value of τ can be modified by `seed.rand`. By adjusting `rho`, the user can also tune the ratio between train and validation. The split function outputs the predicted values and prediction intervals for every component at `x0`.

```
conformal.multidim.split = function(x,y, x0, train.fun, predict.fun, alpha=0.1,
                                    split=NULL, seed=FALSE, randomized=FALSE,seed.rand=FALSE,
                                    verbose=FALSE, rho=0.5,
                                    score="l2",s.type="st-dev", mad.train.fun = NULL,
                                    mad.predict.fun = NULL)
```

In the jackknife+ prediction function, the LOO residuals and n models are computed to obtain fitted values at each point in `x0`. Accordingly, an extended quantile is obtained by using the non-conformity measure in (14) and used to build the lower and upper prediction intervals for every component, which will be returned as output.

```
conformal.multidim.jackplus = function(x,y,x0, train.fun, predict.fun, alpha=0.1)
```

Multi split conformal prediction is the last prediction function, where `B` is the number of times the split conformal function is run. As explained in subsection 2.5, `lambda` controls the smoothing for roughness penalization, while the argument `tau` controls the miscoverage level for the split conformal functions as well as how many points are considered by the extended quantile in (15). `B` is set to 100 by default, while `tau` is set to 0.1. The output is identical to `conformal.multidim.jackplus`.

```
conformal.multidim.msplit = function(x,y, x0, train.fun, predict.fun, alpha=0.1,
                                     split=NULL, seed=F, randomized=F,seed.rand=F,
                                     verbose=F, rho=NULL,score = "max",
                                     s.type = "st-dev",B=100,lambda=0,
                                     tau = 0.1 ,mad.train.fun = NULL,
                                     mad.predict.fun = NULL)
```

The results of a prediction method are plotted using the function `plot_multidim` that takes as input `out`, the output of the prediction method, and a logical `same.scale` that forces the y-axes to utilize the same scale for each component. The function depends on the packages **ggplot2**, **gridExtra** and **hrbrthemes**. Specifically, if we pass the output of full conformal, it plots a heatmap for each test point in `x0`. The color intensifies whenever the estimated p-value increases. Conversely, if the input is the result of split conformal, multi split conformal or jackknife+, then it plots confidence intervals for each observation by pairing a component for the x value and a component for the y value.

```
plot_multidim=function(out,same.scale=FALSE)
```

4. conformalInference.fd : structure and functionality

conformalInference.fd provides conformal prediction when the response variable y is a multivariate functional datum. In the same manner as **conformalInference.multi**, a table of all the available functions is shown in Table 2. A number of helper functions are not present in the table, including the `check` functions, which verify the type correctness of input variables, `convert2data` and `fun2data`, which convert functional inputs (of type *fda*, *fData* or *mfData*) to a list of points, `table2list`, which trasforms matrices to lists.
Moreover, this package also shares the structure with its multivariate counterpart: it is organized into regression methods and prediction functions.

Function	Description
`concurrent`	Build concurrent regression model
`conformal.fun.jackplus`	Computes Jackknife+ prediction sets
`conformal.fun.split`	Computes Split Conformal prediction sets
`conformal.fun.msplit`	Computes Multi Split Conformal prediction sets
`mean_lists`	Build regression method with mean
`plot_fun`	Plot the output prediction methods

Table 2: List of the main functions in **conformalInference.fd**. Helpers functions are excluded.

4.1. Regression methods

Because users may require novel regression models in the future, this package allows them to build their own methods and pass them as arguments to the prediction function. For simpler analysis we just implemented two elementary models for functional regression: the *mean model*, where y is modeled as the mean of the of the observed responses, and a *concurrent regression model*, with the form:

$$y_k(s) = \beta_0(s) + \sum_{i=1}^{p} \beta_i(s) x_i(s) + \epsilon(s) \quad k = 1, ..., q$$

For this latter model, the evaluation grid for function $x \in \prod_{j=1}^{p} L^{\infty}(\tau_j)$ and for $y \in \prod_{j=1}^{q} L^{\infty}(\tau_j)$ must be the same. These regression functions will return a list containing two functions: `train.fun`, which requires as input the lists `x`,`t`, and `y`, and `pred.fun`, which takes as input the test values `x0` and the output of `train.fun`.

4.2. Prediction methods

When dealing with functional data, the input structure tends to be quite complex. To simplify the input, we developed `convert2data`, a function that converts functional data types into lists of pointwise evaluations. The prediction functions make use of the following inputs: `x`, the regressors, `t_x`, the grid over which `x` is evaluated, `y`, the responses, `x0`, the test values, `t_y`, the grid over which `y` is evaluated. `t_x` and `t_y` are lists (length p and q respectively) of vectors. As previously mentioned, it is possible to define `x`,`y` and `x0` as functional data of types fD, $fData$ or $mfData$ (the first type defined in package **fda** and the other two defined in package **roahd**), or to specify them as lists of vectors: the external list describes the observation, the inner one defines the components, while the final vector contains the punctual evaluation of an observation on a specific component (the number of evaluated points may differ). To be more clear:

- $x \to$ list (length n) of lists (length p) of vectors (arbitrary length)
- $y \to$ list (length n) of lists (length q) of vectors (arbitrary length)
- $x_0 \to$ list (length n_0) of lists (length p) of vectors (arbitrary length)

Additionally, the non-conformity score is set to the one presented in (1) and the miscoverage probability of the prediction is $\alpha \in (0, 1)$. The division into train and validation is governed by the arguments `splits`, the vector of indices in training set, `seed`, i.e. the seed for the random engine, and $\rho \in (0, 1)$, which represents the split proportion among training and validation sets. Notice `splits` takes precedence over the other two parameters. One can select the randomized version of the algorithm with `randomize`, while the smoothness value is sampled from an $\mathcal{U}_{[0,1]}$ using `seed.rand`. The user may select a `verbose` version which prints intermediate processes as they occur. With `s.type`, the user has a choice of three modulation methods: *identity*, *st-dev*, and *alpha-max*, that are described in subsection 2.1. All the methods return as output a list containing the evaluation grid for the response `t`, the lower and upper bounds `lo` and `up`. In particular the bounds are lists (of length n_0) of lists (of length p) of vectors. We again used **future.apply** for minimizing computation times, by parallelising the code.

First, we implement the split conformal function. This function generates, as well as the classical outputs, also the predicted values at `x0`. When no value is passed in for `x0`, then the function will use the mean as regression method and output the prediction bands at the validation set. Following Algorithm 2:

```
conformal.fun.split = function(x, t_x, y,t_y, x0, train.fun, predict.fun, alpha=0.1,
                               split=NULL, seed=FALSE, randomized=FALSE,seed.rand=FALSE,
                                  verbose=FALSE, rho=0.5,s.type="st-dev")
```

The second implemented method is jackknife+. Here, we compute the LOO residuals and n models, removing one observation at a time. As described in Algorithm 3, we use the extended quantile defined in (15) to compute the level set of the non-conformity score. The level set contains all the functional data points needed to build a prediction set. The last step involves computing an axis-aligned bounding box (AABB) to determine a predicted set using the functional bands.

```
conformal.fun.jackplus = function(x,t_x, y,t_y, x0, train.fun, predict.fun, alpha=0.1)
```

Alternatively, the multi split conformal function runs `B` times `conformal.fun.split` and joins the B prediction regions into one with the quantile formula in (15), selecting the $2\tau B$ most conformal bands. Lastly, as with jackknife+, the AABB is returned. We recommend you tune the value of `tau` for the particular problem at hand. Our default value is 0.5. `lambda` controls smoothing in the roughness penalization loss. Further details can be found in subsection 2.5.

```
conformal.fun.msplit = function(x,t_x, y,t_y, x0, train.fun, predict.fun, alpha=0.1,
                                split=NULL, seed=FALSE, randomized=FALSE,seed.rand=FALSE,
                                verbose=FALSE, rho=NULL,s.type="st-dev",B=50,lambda=0,
                                tau = 0.5)
```

Our last effort was to design a versatile plot function that takes as input `out`, the output of one of the prediction methods in the package, `y0`, a list composed by the true values of the response at `x0`, `ylab`, a string containing the label for the y-axis, `titles`, the title for the plot, `date`, a vector of date strings (for the date formal look at section 5), `ylim`, the vector of limits for the y-axis and `fillc`, the fill color for the prediction region. With the help of the packages **ggplot2** and **ggpubr**, we can plot all the q dimensions of the response in a single page.

```
plot_fun=function(out,y0=NULL,ylab=NULL,titles=NULL,date=NULL,ylim=NULL,fillc="red")
```

5. Example

As a case study, we examined the BikeMi data, a mobility dataset that tracks all bike rentals for the BikeMi service active in the city of Milan. As in [26], we analyzed 41 days, from the 25$^{\text{th}}$ of January 2016 to the 6$^{\text{th}}$ of March 2016 (excluding the 25$^{\text{th}}$ of February), and all data consisted of time and locations of departures and arrivals of bikes. For simplicity, we are considering only the flow from Duomo district to itself as the functional response, while temperature, humidity, and date are used as regressors.
As the dataset was originally designed for functional modelling, when restricting to the multivariate case some adjustments were required: as response, we used the total number of departures and arrivals from Duomo for every day, while the covariates were the daily averages of the original regressors. Thus, we obtained 41 observations. In short the model is the following:

$$y_i^k = \beta_0^k + \beta_{we}^k x_{we,i} + \beta_{rain}^k x_{rain,i} + \beta_{temp}^k x_{dtemp,i} + \beta_{we\_rain}^k x_{we,i} x_{rain,i} + \epsilon_i^k \qquad k = 1, 2 \;\; i = 1, ..., 41 \tag{19}$$

where k=1 represents the trips starting from Duomo, while k=2 the trips ending in Duomo. $x_{we,i}$ is a logical equal to 1 if day i is weekend. $x_{rain,i}$ is the amount of rain (in mm) on day i at time t. $x_{dtemp,i}$ is the difference from the average daily temperature of the period. $x_{we,i}x_{rain,i}$ is an interaction term between weekend and rain. In practice, we can simply retrieve the data, integrated into the package, choose a test point (March the 6$^{\text{th}}$), build the necessary structures, and pick a linear model as regression method with the following code:

```
library(conformalInference.multi)
data("bikeMi")
n<-nrow(bikeMi)
y<-as.matrix(bikeMi[1:(n-1),1:2],nrow=(n-1))
x<-as.matrix(bikeMi[1:(n-1),-c(1:2)],nrow=(n-1))
y0<-as.matrix(bikeMi[n,1:2],nrow=1)
x0<-as.matrix(bikeMi[n,-c(1:2)],nrow=1)
fun=lm_multi()
```

For prediction we employed all the methods presented in subsection 3.2, with their default values. The only exceptions were `num.grid.pts.dim`, equal to 300 for visualization purposes, the seed in `conformal.multidim.split` and the modulation function chosen (*alpha-max*). All the methods should yield 90% percent valid prediction regions. Additionally, in Figure 1 we plotted the full conformal prediction region as well as the produced prediction regions.

```
full<-conformal.multidim.full(x,y,x0, fun$train, fun$predict,num.grid.pts.dim = 300)
plot_multidim(full)
first<-conformal.multidim.split(x,y,x0, fun$train, fun$predict,seed=123,s.type = "alpha-max")
second<-conformal.multidim.msplit(x,y,x0, fun$train, fun$predict)
third<-conformal.multidim.jackplus(x,y,x0, fun$train, fun$predict)
```

type	**start**	**end**
split	(-2726.59 , 4650.74)	(-1835.98 , 7666.19)
msplit	(-2204.19 , 4429.78)	(-1816.27 , 5177.04)
jack	(-2582.12 , 5371.45)	(-2221.4 , 5970.2)

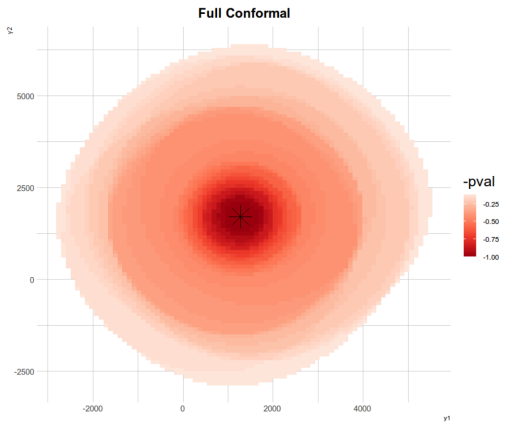


Figure 1: On the left the prediction regions for BikeMi dataset on the 6$^{\text{th}}$ of March 2016, while on the right the heatmap of multivariate full conformal prediction region.

type	**coverage**	**avg area**	**avg time (s)**
full	0.90	$5.68 * 10^7$	35.3
split	0.98	$3.46 * 10^8$	0.01
msplit	0.90	$4.95 * 10^7$	1.80
jack	0.93	$6.56 * 10^7$	1.96

Table 3: Mean coverage, region are and computation time for the multivariate case with BikeMi data.

To determine whether the methods are effective, we extracted each observation $(x_0, y_0) := (x_i, y_i)$ $i = 1, ..., n$ from the training set, constructed the prediction set at x_0, and tested whether the extracted value y_0 was included in the prediction set. We then averaged the coverage, the mean area, and the computation times across all 41 prediction regions: the results are shown in Table 3.
Specifically, one may observe that split conformal method tends to overcover and to produce wider prediction regions, despite being extremely fast. In contrast, the full conformal method generates extremely accurate predictions, but is extremely slow compared to the competition. Ultimately, we found the jackknife+ and multi split extensions to be well-behaved, with reasonable computation times and wide enough regions, while still achieving the 90% coverage target.
In the functional case we considered the original model defined in [26]. Here there are two important differences with respect to (19): the regressors and the responses are time-dependent and the bivariate response is the logarithm of the number of trips started (y^1) and ended (y^2) in the Duomo district at time t. The evaluation grid has 90 time steps and is based on a time window between 7.00 A.M. and 1.00 A.M. The model is as follows:

$$\begin{aligned} log(y_i^k(t)) = \beta_0^k(t) + \beta_{we}^k(t)x_{we,i}(t) + \beta_{rain}^k(t)x_{rain,i}(t) + \beta_{temp}^k(t)x_{dtemp,i}(t) + \\ \beta_{we\_rain}^k(t)x_{we,i}(t)x_{rain,i}(t) + \epsilon_i^k(t) \;\; k = 1, 2 \;\; i = 1, ..., 41 \end{aligned} \tag{20}$$

As the data is already available within the package, we can simply import it. We divide the data into train and test (for test, just consider day 41), build the evaluation grid for y with `t_y` and construct an hour vector for visualization. Finally we select as regression method the concurrent model.

```
library(conformalInference.fd)
data("bike_log")
data("bike_regressors")
x<-bike_regressors[1:40]
x0<-bike_regressors[41]
y<-bike_log[1:40]
y0<-bike_log[41]
t_y<-list(1:length(yb[[1]][[1]]),1:length(yb[[1]][[1]]))
hour_test<-seq(from=as.POSIXct("2016-03-06 07:01:30", tz="CET"),
               to=as.POSIXct("2016-03-07 00:58:30", tz="CET"), length.out=90)
fun<-concurrent()
```

The next step is the actual computation of the bands. Note that for the split conformal method we explicitly selected a seed, allowing repeatability. With the multi split conformal and jackknife+ methods, we chose as number of replication 180 and as τ the value 0.2.

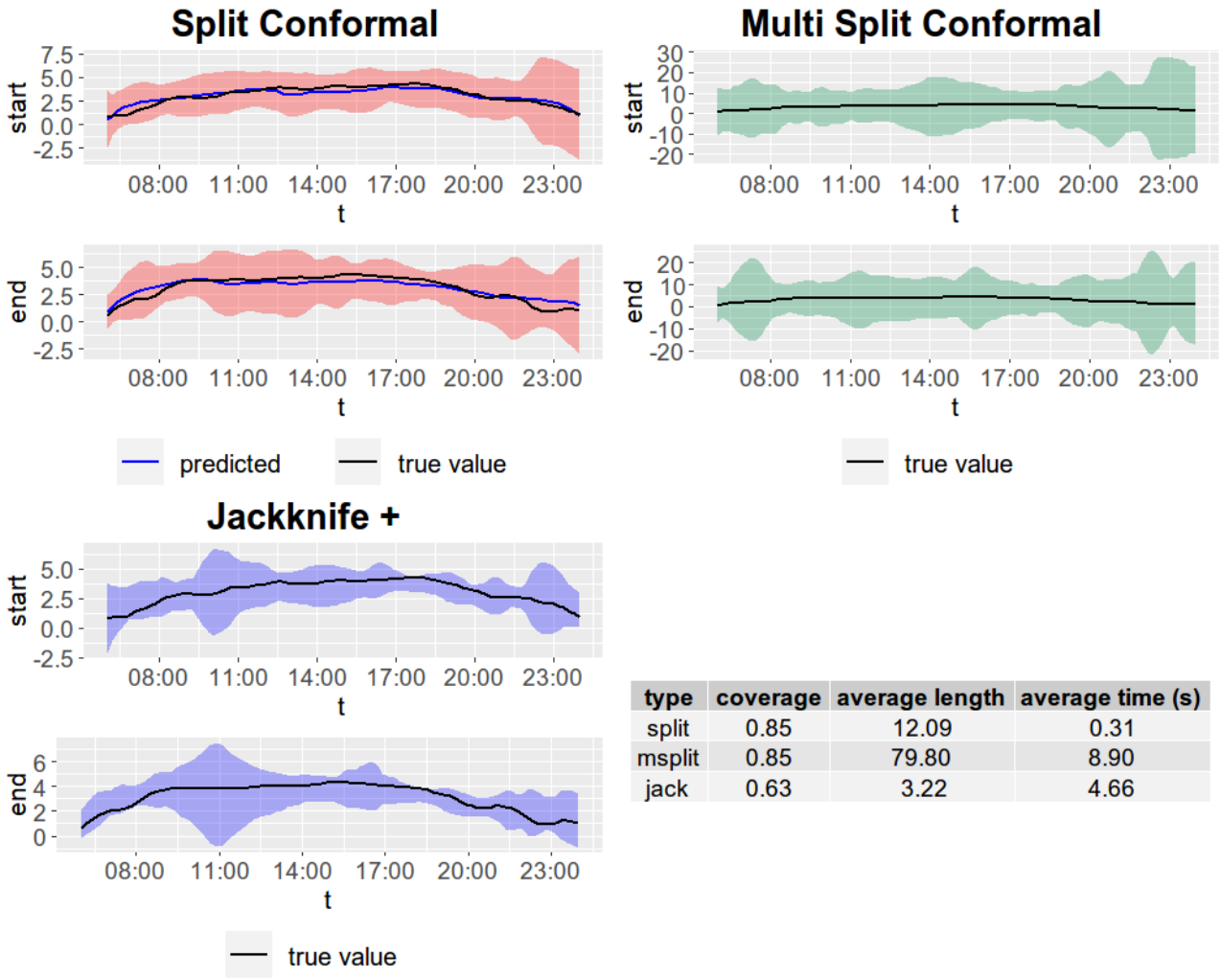
```
first<-conformal.fun.split(x,t_x=NULL,y,t_y,x0,fun$train,fun$predict,seed=1234568)
second<-conformal.fun.msplit(x,t_x=NULL,y,t_y,x0,fun$train,fun$predict,B=180,
tau=0.2)
third<-conformal.fun.jackplus(x,t_x=NULL,y,t_y,x0,fun$train,fun$predict)
```

As a way to display the results we used the `plot_fun` function, where we provided the y-axes labels, the titles, the hour vector, the true response at x_0, and the fill color as input. Figure 2 shows the plots that were produced.

```
plot_fun(first,ylab=c("start","end"),titles="Split Conformal",
         date=list(hour_test,hour_test),y0=y0)
plot_fun(second,ylab=c("start","end"),titles="Multi Split Conformal",
         date=list(hour_test,hour_test),y0=y0,fill="springgreen4")
plot_fun(third,ylab=c("start","end"),titles="Jackknife +",
         date=list(hour_test,hour_test),y0=y0,fill="blue")
```

The final step in evaluating the predictions was to extract one observation from the training set iteratively and use it as a test value. Subsequently, we averaged the coverage and the mean length of the bands and the

computation times. The results are shown in Figure 2. The multi split conformal method appears to construct wider bands than its simpler counterpart (split conformal), while jackknife+ does not seem as well suited to the analysis, providing extremely compact prediction regions and sacrificing coverage. Overall, the split conformal method is the most suitable for this study, since it provides adequate coverage at a minimal computation time and returns reasonable sized bands. However, the split procedure is subject to randomness.



type	coverage	average length	average time (s)
split	0.85	12.09	0.31
msplit	0.85	79.80	8.90
jack	0.63	3.22	4.66

Figure 2: Output of `plot_fun` for the three prediction methods. On the bottom right there are the coverage, average length and computation times of the prediction bands.

6. Conclusions

The article recaps the main concepts of Conformal Prediction theory and proposes extensions to the multivariate and functional frameworks of multi split conformal and jackknife+ prediction methods. Next, the structure and the main functions of **conformalInference.multi** and **conformalInference.fd** are extensively discussed. The last section of the paper presents a case study using mobility data collected by BikeMi and compares the prediction methods on the basis of three factors: the size of the prediction regions, the computation time, and the empirical coverage.

Therefore, we have bridged the gap between R and other programming languages through the introduction of conformal inference tools for regression in multivariate and functional contexts, and as a result shed light on how versatile as well as effective these methods can be.

We envision two future directions for our work. To begin with, in accordance with its author, we would like to submit **conformalInference** for publication on CRAN, which would enrich the pool of distribution-free prediction methods available to R users. Secondarily, we would be interested in expanding on the work presented in [7], to explore conformal inference prediction tools in time series analysis.

References

[1] Baptiste Auguie. *gridExtra: Miscellaneous Functions for "Grid" Graphics*, 2017. R package version 2.3.

[2] Rina Barber, Emmanuel Candès, Aaditya Ramdas, and Ryan Tibshirani. Predictive inference with the jackknife+. *Annals of Statistics*, 49(1):486–507, 2021.

[3] Henrik Bengtsson. *future: Unified Parallel and Distributed Processing in R for Everyone*, 2021. R package version 1.23.0.

[4] Henrik Bengtsson. *future.apply: Apply Function to Elements in Parallel using Futures*, 2021. R package version 1.8.1.

[5] Wenyu Chen, Zhaokai Wang, Wooseok Ha, and Rina Foygel Barber. Trimmed conformal prediction for high-dimensional models, 2016.

[6] Simena Dinas and Jose Bañón. A literature review of bounding volumes hierarchy focused on collision detection. *Ingeniería y Competitividad*, 17:49 – 62, 06 2015.

[7] Jacopo Diquigiovanni, Matteo Fontana, and Simone Vantini. Distribution-free prediction bands for multivariate functional time series: an application to the italian gas market, 07 2021.

[8] Jacopo Diquigiovanni, Matteo Fontana, and Simone Vantini. The importance of being a band: Finite-sample exact distribution-free prediction sets for functional data, 2021.

[9] Jacopo Diquigiovanni, Matteo Fontana, and Simone Vantini. Conformal prediction bands for multivariate functional data. *Journal of Multivariate Analysis*, 189:104879, 2022.

[10] Jonas Fassbender. libconform v0.1.0: a python library for conformal prediction, 2019.

[11] Jerome Friedman, Trevor Hastie, Rob Tibshirani, Balasubramanian Narasimhan, Kenneth Tay, Noah Simon, and James Yang. *glmnet: Lasso and Elastic-Net Regularized Generalized Linear Models*, 2021. R package version 4.1-3.

[12] Chirag Gupta, Arun K. Kuchibhotla, and Aaditya Ramdas. Nested conformal prediction and quantile out-of-bag ensemble methods. *Pattern Recognition*, page 108496, 2021.

[13] Francesca Ieva, Anna Maria Paganoni, Juan Romo, and Nicholas Tarabelloni. roahd Package: Robust Analysis of High Dimensional Data. *The R Journal*, 11(2):291–307, 2019.

[14] Alboukadel Kassambara. *ggpubr: 'ggplot2' Based Publication Ready Plots*, 2020. R package version 0.4.0.

[15] Danijel Kivaranovic and Hannes Leeb. A (tight) upper bound for the length of confidence intervals with conditional coverage. *arXiv: Methodology*, 2020.

[16] Jing Lei, Max G'Sell, Alessandro Rinaldo, Ryan Tibshirani, and Larry Wasserman. Distribution-free predictive inference for regression. *Journal of the American Statistical Association*, 113(523):1094–1111, 2018.

[17] Henrik Linusson. nonconformist, 2017. Python package version 2.1.0 (on GitHub).

[18] Nicolai Meinshausen. *quantregForest: Quantile Regression Forests*, 2017. R package version 1.3-7.

[19] J. O. Ramsay, Spencer Graves, and Giles Hooker. *fda: Functional Data Analysis*, 2021. R package version 5.5.1.

[20] Yaniv Romano, Evan Patterson, and Emmanuel J. Candès. Conformalized quantile regression. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[21] Bob Rudis. *hrbrthemes: Additional Themes, Theme Components and Utilities for 'ggplot2'*, 2020. R package version 0.8.0.

[22] Matteo Sesia and Emmanuel J. Candès. A comparison of some conformal quantile regression methods. *Stat*, 9(1):e261, 2020.

[23] Aldo Solari and Vera Djordjilović. Multi split conformal prediction. *Statistics Probability Letters*, 184:109395, 2022.

[24] Ryan Tibshirani. conformalinference: Tools for conformal inference in regression, 2019. R package version 1.1.0 (on GitHub).

[25] Ryan J. Tibshirani, Rina Foygel Barber, Emmanuel J. Candès, and Aaditya Ramdas. Conformal prediction under covariate shift. In *NeurIPS*, 2019.

[26] Agostino Torti, Alessia Pini, and Simone Vantini. Modelling time-varying mobility flows using function-on-function regression: Analysis of a bike sharing system in the city of milan. *Journal of the Royal Statistical Society.Series C: Applied Statistics*, 70(1):226–247, 11 2020.

[27] Vladimir Vovk, Alexander Gammerman, and Glenn Shafer. *Algorithmic learning in a random world*. Springer Science Business Media, 2005.

[28] Hadley Wickham, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus Wilke, Kara Woo, Hiroaki Yutani, and Dewey Dunnington. *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*, 2021. R package version 3.3.5.

[29] Gianluca Zeni, Matteo Fontana, and Simone Vantini. Conformal prediction: a unified review of theory and new challenges. *ArXiv*, abs/2005.07972, 2020.

A. Appendix Univariate Response

This thesis covers another topic: Conformal Prediction in the univariate response case. As mentioned in the introduction, R already has a well-designed package for conformal prediction in this framework: **conformalInference**. It was created by Professor Ryan Tibshirani of the Statistics and Machine Learning Department at Carnegie Mellon University, and contains a set of effective tools for conformal prediction. Nevertheless, we felt there were a few innovative methods missing. Due to this, we approached him and his team and proposed our implementations. Beyond minor tweaks to improve the package performance (such wide use of parallelisation with **future.apply**), we implemented three major prediction methods: Jackknife+, Multi Split Conformal and Conformalized Quantile Regression.
We should note that this work is not discussed in the main body of the thesis, as this package is not yet published on CRAN, unlike its multivariate and functional counterparts. Thus, we decided to illustrate our contributions in the appendix.
The package shares the same philosophy as **conformalInference.multi** and **conformalInference.fd**: the training and prediction functions are passed as input by the user (more details can be found in subsection 3.2). All the functions take as input `x`, the matrix of covariates, `y`, the vector of responses, `x0`, the matrix of test points, `alpha`, the miscoverage probability, the mad functions, needed to locally scale the residuals (go to subsection 2 for more details), and a logical `verbose` to display intermediate processes. Furthermore, all the prediction functions return the vectors of lower and upper bounds.

A.1. Jackknife+

As described in subsection 2.4, Jackknife+ is based upon the concept of leave-one-out residuals and, in contrast with the original Jackknife method, it does not provide symmetric intervals. It leads to valid $1 - 2\alpha$ prediction intervals. The method is inherently very demanding from a computational point of view. It was, then, crucial to quicken the computation of LOO residuals (e.g. using the Sherman-Morrison formula for linear models) and to employ parallel computation whenever possible. Additionally, whenever the user lacks **future.apply**, a sequentially apply is run, without interrupting abruptly the function.
We modified the already existing `conformal.pred.jack` function, which originally handled only the classic jackknife prediction method. Consequently, when the user sets the parameter `plus = TRUE` it performs jackknife+, otherwise it proceeds with jackknife. Moreover the user may pass `special.fun`, a function to compute LOO residuals with shortcuts. The function is structured as follows:

```
conformal.pred.jack = function(x, y, x0, train.fun, predict.fun, alpha=0.1,
  special.fun=NULL, mad.train.fun=NULL, mad.predict.fun=NULL,
  plus=TRUE,verbose=FALSE)
```

A.2. Multi split conformal

The second integration involved the Multi Split conformal method, which we have already described in subsection 2.5. Using this approach, multiple split conformal prediction intervals are joined, preventing different results

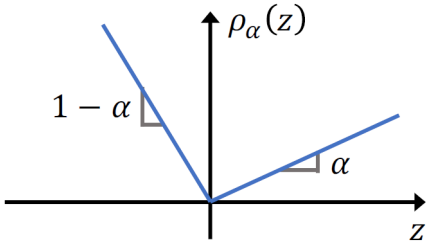


Figure 3: Pinball loss for quantile regression. Set $z = y - \hat{y}$

from different runs of the function. This method leads to valid $1-\alpha$ prediction intervals. The structure is the same as that defined in Algorithm 4. B is the number of times the split conformal algorithm is run. A crucial role is also played by the parameter τ. It determines how many split conformal intervals a point must be included to become a part of the final prediction. According to the sensitivity analysis performed in [23], we set it as default as $1 - \frac{B+1}{2B}$. ρ is a vector of length B, containing the split proportion between training and validation sets during each split conformal run. Alternatively, a matrix of seeds or a vector of splits can be used to control the splitting for each run of the split conformal algorithm. In addition, the user can provide a smoothing value (`lambda`) and set of weights (`w`) to account for a potential shift in the distribution of input values between training and testing (this problem is known as *covariate shift*[6]).

```
conformal.pred.msplit = function(x, y, x0,train.fun, predict.fun, alpha=0.1,
                                 rho = rep(0.5,B), w=NULL,
                                 mad.train.fun=NULL, mad.predict.fun=NULL,
                                 split=NULL, seed=NULL,
                                 verbose=FALSE,
                                 B=50,
                                 lambda = 0.5,
                                 tau = 1-(B+1)/(2*B))
```

A.3. Conformalized quantile regression

Conformalized Quantile Regression (**CQR**) is an innovative prediction method, introduced in [20], which combines the theory of quantile regression and the theory of conformal prediction. Quantile regression is a family of methods to estimate the α^{th} quantile function from a set $\{(x_1, y_1), ..., (x_n, y_n)\}$ as:

$$q_\alpha(x) = inf\{y \in \mathbb{R} : F(y|X = x) \geq \alpha\} \tag{21}$$

Then, we may introduce the parameter γ and consequently define $\alpha_{lo} := \gamma$ and $\alpha_{up} := 1 - \gamma$. While in the practical implementations the user may freely select the value of γ, in the theoretical study we will fix $\gamma = \alpha/2$. Therefore, by construction, the quantile function has a vital property:

$$\mathbb{P}(Y \in [q_{\alpha_{lo}}(X), q_{\alpha_{up}}(x)] \,|X = x) \geq 1 - \alpha \tag{22}$$

Essentially, the estimation of $q_\alpha(x)$ may be viewed as an optimization problem with the form:

$$\hat{q}_\alpha(x) = f(x; \hat{\theta}) \qquad \hat{\theta} = \arg\min_\theta \frac{1}{n} \sum_{i=1}^{n} \rho_\alpha(Y_i, f(X_i; \theta)) + \epsilon(\theta) \tag{23}$$

where $f(x; \theta)$ is the quantile regression function, while $\rho_\alpha(Y_i, f(X_i; \theta))$ is the so-called *pinball loss*. This loss, displayed in Figure 3, is formally defined as:

$$\rho_\alpha(y, \hat{y}) = \begin{cases} \alpha(y - \hat{y}) & \text{if } (y - \hat{y}) > 0 \\ (1 - \alpha)(\hat{y} - y) & \text{elsewhere} \end{cases} \tag{24}$$

Since the problem formulation in (23) is extremely general, one may model $\hat{q}_{alpha}(x)$ with various algorithms (e.g. using lasso regression or neural networks). As our base model we chose quantile regression forests from the package **quantregForest**. This tool asks the user for the number of threads to use for parallel computation

[6] A detailed description of covariate shift can be found in [25].

with the parameter `nthreads`.
Mirroring equation (22) we can suggest as a naive prediction interval:

$$C_{naive}(X_i) = [\hat{q}_{\alpha_{lo}}(X_i), \hat{q}_{\alpha_{up}}(X_i)] \tag{25}$$

While this intervals shows a tendency to undercover, we can still use it as reference to construct a suitable non-conformity score:

$$R_i := max(\hat{q}_{\alpha_{lo}}(X_i) - Y_i, Y_i - \hat{q}_{\alpha_{up}}(X_i)) \tag{26}$$

Its interpretation is fairly straightforward. When Y_i is below the estimated lower bound $\hat{q}_{\alpha_{lo}}(X_i)$, their distance is returned. The converse holds when Y_i is above the estimated upper bound. Finally, when Y_i belongs to the interval $C_{naive}(X_i)$, the larger distance between Y_i and the two endpoints is returned. To avoid confusion, the non-conformity measure applied in equation (26) is called *classical*.
A more robust non-conformity measure takes into account the distance of the quantile functions from the median. The *median* option, taken from [15], behaves as follows:

$$R_i := max\left(\frac{\hat{q}_{\alpha_{lo}}(X_i) - Y_i}{\hat{q}_{0.5}(X_i) - \hat{q}_{\alpha_{lo}}(X_i)}, \frac{Y_i - \hat{q}_{\alpha_{up}}(X_i)}{\hat{q}_{\alpha_{up}}(X_i) - \hat{q}_{0.5}(X_i)}\right) \tag{27}$$

There exists a third alternative: the *scaled* version. As described in [22], it scales the function in equation (26) by the length of the naive prediction interval $C_{naive}(X_i)$:

$$R_i := max\left(\frac{\hat{q}_{\alpha_{lo}}(X_i) - Y_i}{\hat{q}_{\alpha_{up}}(X_i) - \hat{q}_{\alpha_{lo}}(X_i)}, \frac{Y_i - \hat{q}_{\alpha_{up}}(X_i)}{\hat{q}_{\alpha_{up}}(X_i) - \hat{q}_{\alpha_{lo}}(X_i)}\right) \tag{28}$$

From here on we will distinguish among full CQR and split CQR. The full conformal alternative estimates $\hat{q}_{\alpha_{lo}}$ and $\hat{q}_{\alpha_{up}}$ considering all the observations in the training set, whereas the split conformal one exploits only observations belonging to the training set $\mathcal{I}_1$.

We have enough elements to create our version of full conformal for CQR. As it is still a full conformal function, the coverage guarantee at level $1 - \alpha$ holds. Further description of the essential paramters for full conformal may be found in subsection 2.2. We should keep in mind all the considerations we made about the value of `gamma`. Moreover, we also included a string `method` to choose among the three non-conformity measures, and covariate shift is tackled once again by the parameter `w`. The `conformal.quant` function is presented below:

```
conformal.quant = function(x, y, x0, method="scaled", nthreads = 8,
                           alpha=0.1,
                           gamma=alpha/2, verbose=FALSE,w=NULL,
                           mad.train.fun=NULL,
                           mad.predict.fun=NULL, num.grid.pts=25, grid.factor=1.25)
```

The split conformal version of CQR requires two additional elements. Firstly, we call k the $(1-\alpha)(1+1/|\mathcal{I}_2|)^{th}$ empirical quantile of $\{R_i : i \in \mathcal{I}_2\}$. Secondly, for any non-conformity measure, we must define the correspondent prediction intervals:

$$\textit{classical} \qquad C(x) = [\hat{q}_{\alpha_{lo}}(x) - k, \hat{q}_{\alpha_{up}}(x) + k] \tag{29}$$

$$\textit{median} \qquad C(x) = (1+k)[\hat{q}_{\alpha_{lo}}(x), \hat{q}_{\alpha_{up}}(x)] - k\hat{q}_{1/2}(x) \tag{30}$$

$$\textit{scaled} \qquad C(x) = [\hat{q}_{\alpha_{lo}}(x), \hat{q}_{\alpha_{up}}(x)] \pm k(\hat{q}_{\alpha_{up}}(x) - \hat{q}_{\alpha_{lo}}(x)) \tag{31}$$

While the *classical* case does not apply any local scaling, forcing all intervals to the same length $2k$, both the *median* and *scaled* cases resize the intervals by a certain factor. As shown in [20], split CQR satisfies the marginal, distribution-free coverage guarantee at level $1 - \alpha$. Finally, we can present its implementation:

```
conformal.quant.split = function(x, y, x0, method="scaled", nthreads = 8,
                                 alpha=0.1,gamma=alpha/2, split=NULL,
                                 seed=NULL, verbose=FALSE)
```

Abstract in lingua italiana

I metodi appartenenti alla famiglia della Conformal Prediction generano insiemi di previsione non parametrici per il caso regressivo, a condizione di avere dati regressivi *i.i.d.* Sebbene vi siano dei pacchetti di R che trattano questi metodi per risposte univariate (**conformalInference**), questo non è il caso per risposte multivariate o funzionali. **conformalInference.multi** e **conformalInference.fd** colmano questa lacuna, implementando metodi quali full conformal, split conformal, e innovative estensioni di jackknife+ e multi split conformal per gestire dati multivariati e funzionali. L'estrema flessibiltà della conformal prediction, che non necessita di alcun modello di regressione specifico, permette all'utente di passare qualsiasi funzione regressiva come input, prendendo spunto dai modelli regressivi già inclusi nei pacchetti.
Inoltre, per visualizzare queste regioni di previsione sono state incluse delle funzioni grafiche. Infine, nell'appendice è presente una serie di contributi al pacchetto di R **conformalInference**, fra cui anche il metodo chiamato Conformalized Quantile Regression.

Parole chiave: Conformal Prediction, dati Multivariati Funzionali, insieme di previsione, Multi Split Conformal, Jackknife+, Conformalized Quantile Regression