Proceedings of Machine Learning Research 266:1–20, 2025 Conformal and Probabilistic Prediction with Applications

online-cp: a Python Package for Online Conformal Prediction, Conformal Predictive Systems and Conformal Test Martingales

Johan Hallberg Szabadváry JOHAN.HALLBERG.SZABADVARY@JU.SE
Tuwe Löfström TUWE.LOFSTROM@JU.SE
Rudy Matela RUDY.MATELA@JU.SE
Jönköping University, Sweden

Editor: Khuong An Nguyen, Zhiyuan Luo, Harris Papadopoulos, Tuwe Löfström, Lars Carlsson and Henrik Boström

Abstract

Conformal prediction (CP) has gained increasing attention in machine learning owing to its ability to provide reliable prediction sets with well-calibrated uncertainty estimates. While most existing CP implementations focus on inductive conformal prediction (ICP), full conformal prediction—also known as online or transductive CP—offers the strongest validity guarantees but has been largely absent from open-source software due to its computational complexity. In this paper, we introduce `online-cp`, a Python package designed for online conformal prediction, conformal predictive systems (CPS), and conformal test martingales. The package implements several online CP algorithms, enabling efficient and principled uncertainty quantification in streaming data scenarios. Additionally, it includes tools for testing the exchangeability assumption by using conformal test martingales. We demonstrate the functionality of `online-cp` through classification and regression examples as well as applications to predictive systems and exchangeability testing. By making online CP methods accessible, `online-cp` provides a foundation for the broader adoption and further development of conformal prediction in real-time machine learning applications.

Keywords: Online conformal prediction, Conformal predictive systems, Conformal test martingales, Machine learning, Uncertainty quantification, Exchangeability testing, Python package

1. Introduction

Conformal predictors emphasise reliability. They evaluate the reliability of their own predictions, such that they are neither over-pessimistic nor over-optimistic. This is achieved by transforming the typical prediction problem of point prediction into set prediction. In its most basic form, conformal prediction (CP) is a method for producing prediction sets that are provably valid, in the sense that they contain the true outcome with at least a user-specified probability. The size of the prediction set reflects the uncertainty. Introduced by Vovk, Gammerman, and Shafer (Vovk et al., 2005, 2022), interest in CP has increased rapidly with the application of machine learning to real-world high-stakes situations, where honest and reliable predictions are essential for safe operation.

Several software packages have been developed with increasing interest in CP. In Python, the most widely used are `crepes` (Boström, 2022) and `MAPIE` (Cordier et al., 2023), and software is also available for R (Gauraha and Spjuth, 2018; Lei et al., 2018) and Julia (Altmeyer and contributors, 2023). Except for the package by Gauraha and Spjuth (2018),

these are all focused on inductive or split CP (ICP), which is by far the most commonly used version of CP. The latest update of `crepes`[1] included semi-online ICP, which iteratively update the calibration set after making each prediction. However, the strongest form of validity requires what is known as full, online, or transductive CP, which operates in the online setting. The, typically, high computational complexity of online CP and its restriction to online learning scenarios mean that open-access software tools have not been prioritised and are not available in the widely used Python language. Another reason may be that while ICP is quite convenient to design using "wrappers" for established ML software packages, such as `scikit-learn` (Pedregosa et al., 2011), online CP typically requires specialised algorithms, particularly for regression.

Our contribution to the open-source software is `online-cp`, a Python package for online conformal prediction, where we have implemented several online algorithms from the monograph Vovk et al. (2022), as well as conformal test martingales to test the important assumption of exchangeability. We hope that the openly available software for online CP will make it convenient to use CP in its natural habitat–the online setting, which is the subject of the next section, where we highlight the difference between inductive and full CP. Section 3 explains how to install and set up everything required to start working with `online-cp`. Examples of conformal classification, regression, and conformal predictive systems (CPS), as well as conformal test martingales, are presented in Section 4. Finally, in Section 5, we summarise the remaining work and extend an open invitation to anyone interested in contributing to further development.

2. Online conformal prediction and test martingales

We give a very brief introduction to the most important aspects of online CP and conformal test martingales. We refer to Vovk et al. (2022) for full treatment of the subject. Readers already familiar with conformal prediction in the online setting may find it convenient to skip this section.

2.1. Conformal prediction in the online setting

In the online setting, Reality outputs successive pairs

$$(x_1, y_1), (x_2, y_2), \ldots \tag{1}$$

called examples. Each example consists of an object $x_i \in \boldsymbol{X}$ and its label $y_i \in \boldsymbol{Y}$, where the measurable spaces $\boldsymbol{X}$ and $\boldsymbol{Y}$ are called the example and label spaces, respectively. We often find it convenient to write $z_i := (x_i, y_i)$, which is an element of the example space $\boldsymbol{Z} := \boldsymbol{X} \times \boldsymbol{Y}$. The standard assumption in CP is that the sequence of examples is drawn from a probability distribution P on $\boldsymbol{Z}^\infty$ that is exchangeable. Intuitively, exchangeability means that the distribution of the examples does not depend on their order; that is, any permutation of the sequence (1) is equally likely. Essentially, this means that the order in which the examples are observed is irrelevant. For details on exchangeability, we refer to Vovk et al. (2022).

1. https://github.com/henrikbostrom/crepes/releases/tag/v0.8.0

We write $\wr z_1, \dots, z_n \int$ for the bag, or multiset, of examples observed at step n. A bag is similar to a set in that the order of elements is irrelevant; however, in contrast to sets, several elements in a bag may be identical. The set of all bags of size n of elements from a measurable set $\boldsymbol{Z}$ is denoted by $\boldsymbol{Z}^{(n)}$.

Formally, a nonconformity measure is a measurable map

$$A : \boldsymbol{Z}^{(*)} \times \boldsymbol{Z} \to \overline{\mathbb{R}}.$$

where $\overline{\mathbb{R}} = \mathbb{R} \cup \{+\infty\}$. It is a way of scoring how strange or unusual an example is as an element of a bag. Given a nonconformity measure A and a bag $\sigma := \wr z_1, \dots, z_n \int$, we can quantify the strangeness of an example z by $\alpha_i := A(\sigma, z_i)$. This on its own does not tell use how unusual A finds z, but the fraction

$$\frac{|\{j = 1, \dots, n : \alpha_j \geq \alpha_i\}|}{n},$$

where n is the number of elements in the bag, indicates how unusual A finds the example z_i in the bag σ. This fraction is called the p-value for z_i.

Formally, the (smoothed) conformal predictor determined by the nonconformity measure A is a measurable function, $\Gamma : (\boldsymbol{X} \times [0,1] \times \boldsymbol{Y})^* \times (\boldsymbol{X} \times [0,1] \times \boldsymbol{Y}) \times (0,1) \to 2^{\boldsymbol{Y}}$, where $2^{\boldsymbol{Y}}$ is the set of all subsets of $\boldsymbol{Y}$. For each significance level $\varepsilon \in (0,1)$, it outputs the prediction set

$$\begin{aligned}
&\Gamma^\varepsilon(x_1, \tau_1, y_1, \dots, x_{n-1}, \tau_{n-1}, y_{n-1}, x_n, \tau_n) \\
&\quad := \left\{ y \in \boldsymbol{Y} : \frac{|\{i = 1, \dots, n : \alpha_i > \alpha_n\}| + \tau_n |\{i = 1, \dots, n : \alpha_i = \alpha_n\}|}{n} > \varepsilon \right\}
\end{aligned}$$

where

$$\begin{aligned}
\alpha_i &:= A(\sigma, (x_i, y_i)), \ i = 1, \dots, n-1 \\
\alpha_n &:= A(\sigma, (x_n, y)), \\
\sigma &:= \wr (x_1, y_1), \dots, (x_{n-1}, y_{n-1}), (x_n, y) \int .
\end{aligned}$$

The numbers τ_i, $i = 1, \dots, n$ are random numbers, independent of everything else (as can be expected if they are generated by a random number generator), and distributed according to the uniform distribution on $[0,1]$. It is also possible to define non-smoothed conformal predictors, which is equivalent to setting $\tau_i = 1$ for $i = 1, \dots, n$. The main property of smoothed conformal predictors is that they are exactly valid. This property is worth formally defining. Let the smoothed conformal predictor Γ process the infinite data sequence

$$\omega = (x_1, y_1, x_2, y_2, \dots)$$

at significance level ε. We say that Γ makes an error at the nth trial if $y_n \notin \Gamma^\varepsilon_n$. More precisely,

$$\operatorname{err}^\varepsilon_n(\Gamma, \omega) := \begin{cases} 1 & \text{if } y_n \notin \Gamma^\varepsilon_n, \\ 0 & \text{otherwise.} \end{cases}$$

The number $\operatorname{err}^\varepsilon_n(\Gamma, \omega)$ is the realisation of a random variable $\operatorname{err}^\varepsilon_n(\Gamma)$. We say that a confidence predictor Γ is exactly valid if, for each ε,

$$\operatorname{err}^\varepsilon_1(\Gamma), \operatorname{err}^\varepsilon_2(\Gamma), \dots$$

is a sequence of independent Bernoulli random variables with parameter ε. In other words, the event of making an error is like getting heads when tossing a biased coin, where the probability of getting heads is always ε. There is also the notion of conservative validity, which is more complicated to state but essentially means that the error sequence is dominated by a sequence of independent Bernoulli variables with parameter ε. For the complete statement, we refer to Vovk et al. (2022). Non-smoothed conformal predictors are conservatively valid.

In the offline setting, it is possible to retain approximate conservative validity, which holds conditional on the calibration set (see Corollary 4.9 in Vovk et al. (2022)); however, the error events are no longer independent.

2.2. Inductive conformal prediction in the offline setting

In the offline setting, the more widely used inductive or split conformal predictors (ICP) are defined as follows. An inductive nonconformity measure is a function $A : \boldsymbol{Z}^* \times \boldsymbol{Z} \to \overline{\mathbb{R}}$. Given a training set of size l, we split it into two parts: the proper training set $z_1, \ldots, z_m$ of size $m < l$ and the calibration set $z_{m+1}, \ldots, z_l$ of size $h := m - l$. For a test object x_i, the prediction set is then

$$\Gamma^\varepsilon(z_1, \ldots, z_l, x_i) := \left\{ y \in \boldsymbol{Y} : \frac{|\{j = m+1, \ldots, l : \alpha_j \geq \alpha_i\}| + 1}{h+1} > \varepsilon \right\},$$

where the nonconformity scores are given by

$$\begin{aligned} \alpha_j &:= A((z_1, \ldots, z_m), z_j), \ j = m+1, \ldots, l \\ \alpha_i &:= A((z_1, \ldots, z_m), (x_i, y)). \end{aligned}$$

The training-conditional validity of ICP, that is, the coverage probability conditioned on the training set, is discussed in Section 4.7.2, of Vovk et al. (2022). It is of the Probably Approximately Correct (PAC) type and involves two parameters: the significance level ε and another parameter $\delta \in (0, 1)$. That is, a set predictor Γ is (ε, δ)-valid if, with a probability of at least $1 - \delta$, the coverage probability of the prediction set is at least $1 - \varepsilon$. A full discussion is beyond the scope of this paper, but it is shown in Vovk et al. (2022) how to construct (ε, δ)-valid ICPs when the calibration set is sufficiently large. Specifically, $\Gamma^{\varepsilon - \sqrt{\frac{\ln 1/\delta}{2h}}}$ is (ε, δ)-valid.

2.3. Conformal test martingales

The exact validity of smoothed conformal predictors relies on the fact that, under the assumption of exchangeability, the p-values associated with the correct label, y_i for $i = 1, 2 \ldots$ are independent and distributed uniformly on $[0, 1]$, by Theorem 11.1 in Vovk et al. (2022). This property can be used to test the exchangeability hypothesis. A martingale is a sequence of non-negative random variables $M_0, M_1, \ldots$ whose conditional expectation remains unchanged.

$$\begin{aligned} M_n &\geq 0 \\ \mathbb{E}[M_{n+1} \mid M_1, \ldots, M_n] &= M_n \ a.s \end{aligned} \tag{2}$$

If we set $M_0 = 1$, we obtain a test martingale, which may be interpreted as representing the capital of a gambler that starts for unit capital and bets on the outcome of events, never

risking bankruptcy. The requirement (2) reflects the fairness of the game: at step $n-1$, the conditional expected value of the gamblers' capital at step n is her current capital. In conformal testing, the gambler bets against the hypothesis that the p-values $p_1, p_2, \ldots$ are independent and distributed uniformly on $[0,1]$. Intuitively, if the p-values actually are distributed according to this hypothesis, she is unlikely to become very rich betting against their uniformity. Exactly how unlikely, is quantified by Ville's inequality (Ville, 1939), which states

$$\mathbb{P}\left\{\exists n : M_n \geq C\right\} \leq 1/C, \ \forall C \geq 1. \tag{3}$$

Testing the exchangeability assumption may thus be done by defining a conformal test martingale and discarding the exchangeability hypothesis at significance $1/C$ if we ever observe that the martingale value exceeds $C > 0$. Several conformal test martingales are described in Vovk et al. (2005, 2022) and more have been suggested by Vovk et al. (2003) and Fedorova et al. (2012a).

3. Installing and importing online-cp

The source code of the `online-cp` python package, together with example notebooks, can be found as follows:

- https://github.com/egonmedhatten/online-cp
- https://pypi.org/project/online-cp/

It is licenced under a permissive BSD-3-Clause licence. This paper details `online-cp` as of version 0.1.0. It is under continuous integration (CI) with a few automated tests and sanity checks.

To install the package from the Python Package Index (PyPI)[2], one types the following at the command prompt of the operating system, assuming that `pip`[3] is already installed:

```
pip install online-cp
```

This instals the latest version of `online-cp` and allows the package to be imported into Python programs and Jupyter notebooks et cetera. The next section illustrates how this can be done using a few examples.

4. Examples

We now consider a few examples of the classes and methods included in `online-cp`. The package is still in the early stage of development, and more methods will be added. At the time of writing, the package exposes the following classes:

- `ConformalClassifier`
  - `ConformalNearestNeighboursClassifier`

2. https://pypi.org/
3. https://pip.pypa.io/

- `ConformalRegressor`
  - `ConformalRidgeRegressor` (basic and studentized as discussed in Vovk et al. (2022))
- `ConformalPredictiveSystem`
  - `RidgePredictionMachine` (basic and studentized)
- `ConformalTestMartingale`
  - `PluginMartingale`
  - `SimpleJumperMartingale`

The methods under development include kernel versions of regressors and conformal predictive systems as well as more conformal test martingales.

Taking inspiration from the online machine learning Python library `river`[4] (Montiel et al., 2021), conformal predictors learn one example at a time, using the `learn_one` method. It is also possible to learn a "batch" of examples by using `learn_many`. If initial training data are available, `learn_initial_training_set` pretrains the CP.

All examples were computed on a laptop with an Intel Core i7-1255U processor (12× logical cores, Performance-core Max Turbo Frequency up to 4.70 GHz, Efficient-core Max Turbo Frequency up to 3.50 GHz), 40GB RAM, running (Linux) Pop!_OS 22.04 LTS.

4.1. Classification

We illustrate the classification procedure in the online setting using a synthetic dataset with object space $\boldsymbol{X} = \mathbb{R}^2$ and label space $\boldsymbol{Y} = \{0, 1, 2\}$. The dataset was generated using the `scikit-learn` function `make_classifcation`. An illustration of the dataset is shown in Figure 1. For example, we use the `ConformalNearestNeighboursClassifier`, with the number of neighbours $k = 1$, to process the dataset. With $k = 1$, as discussed in Chapter 3 (Vovk et al., 2022), the nonconformity measure is

$$A(\{z_1, \ldots, z_l, (x, y)\}, y) = \frac{\min_{i=1,\ldots,l:y_i=y} d(x, x_i)}{\min_{i=1,\ldots,l:y_i\neq y} d(x, x_i)}$$

where d denotes Euclidean distance. If another distance function is required, this can be controlled by passing a keyword in the constructor. If `distance` is compatible with `cdist` in `scipy`[5] (Virtanen et al., 2020), the distance is computed according to the corresponding `scipy` distance. Even more flexibility can be had by setting `distance` to `'custom'`, and passing a custom distance function using the keyword `distance_func`.

All conformal predictors and conformal predictive systems can be evaluated using the `Evaluation` class, which is constructed using the metrics we wish to keep track of during the processing of a dataset. At the time of writing, some of the efficiency criteria listed in Vovk et al. (2022) have been implemented, and users can construct their own evaluation metrics to suit their particular needs. For a classification task, the evaluation object is initialised as follows:

4. https://github.com/online-ml/river
5. https://scipy.org/

```
from online_cp import Evaluation, Err, OF, OE
metrics = Evaluation(err=Err, oe=OE, of=OF)
```

The `Err` class tracks committed cumulative errors. The other keywords allow us to choose which efficiency criteria to keep track of. The `OE` class tracks the observed excess (OE), defined as

$$\frac{1}{n}\sum_{i=1}^{n}\sum_{y\neq y_i} p_i^y,$$

that is, the number of incorrect labels included in the prediction sets. The `OF` class keeps track of the observed fuzziness, which is the sum of the incorrect p-values

$$\frac{1}{n}\sum_{i=1}^{n}|\Gamma_i^\varepsilon \setminus \{y_i\}$$

Both OE and OF are conditionally proper efficiency criteria, as discussed in Chapter 3 of Vovk et al. (2022).

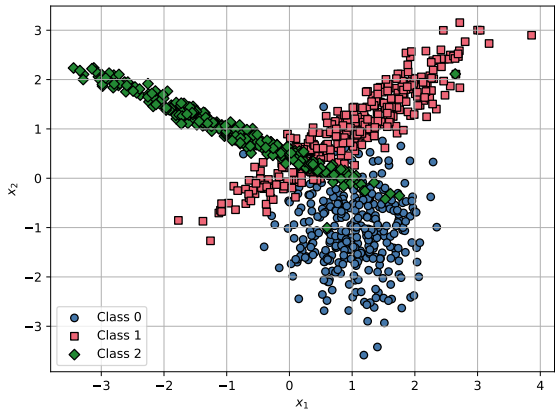


Figure 1: Synthetic dataset for classification. It consists of three classes and 1000 examples.

To process the synthetic dataset, we ran the following code:

```
from online_cp import ConformalNearestNeighboursClassifier
cp = ConformalNearestNeighboursClassifier(k=1, label_space=np.unique(y), epsilon
    =0.1)

for i, (obj, lab) in enumerate(zip(X, y))

    # Make prediction
    Gamma, p_values, D = cp.predict(obj, return_p_values=True, return_update=True)

    # Learn the label
    cp.learn_one(obj, lab, D)

    # Update efficiency criteria
    metrics.update(y=lab, Gamma=Gamma, p_values=p_values)
```

`return_update` returns the updated distance matrix, which can then be reused in `learn_one` to avoid repeating the same computations. The output of `predict` is a `ConformalPredictionSet`, which is represented as a `numpy` array with the included labels. It also has a `size` method, which returns the cardinality of the set and saves the significance level at which it was constructed. The p-values are stored in a dictionary with the label space as keys. To view the results, the command

```
metrics.plot_cumulative_results()
```

produces the plots shown in Figure 2.

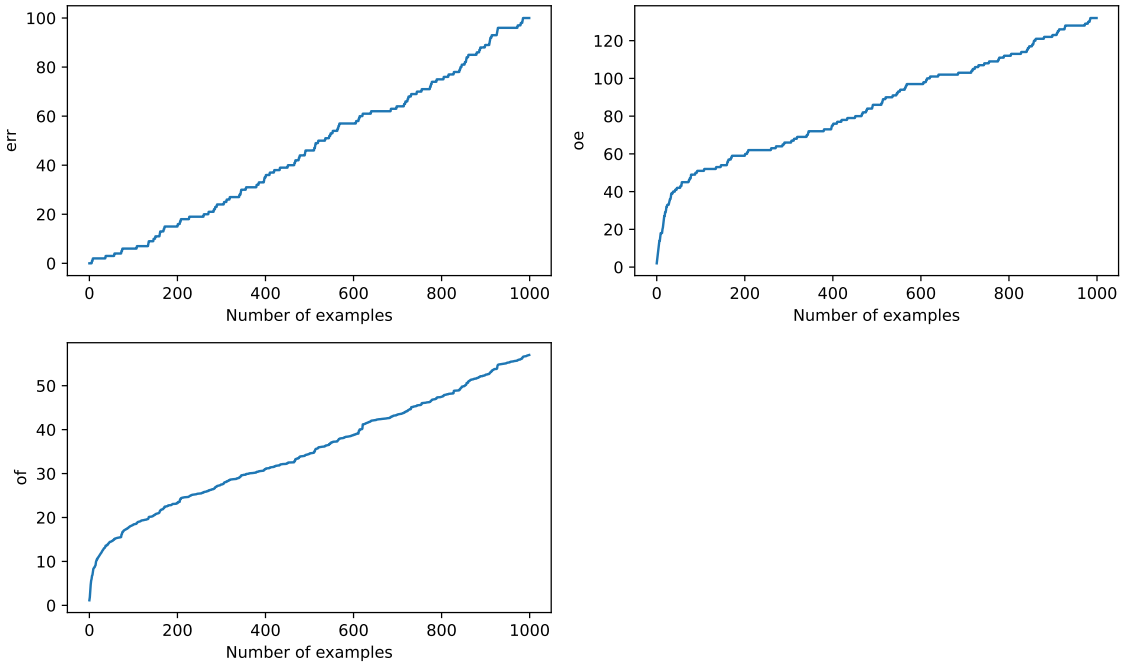


Figure 2: Cumulative results for synthetic classification.

The computational complexity of the conformal nearest neighbour classifier per iteration is dominated by the maintenance and update of the pairwise distance matrix, which scales quadratically with the number of samples n. Specifically, the complexity per iteration is $\mathcal{O}(n^2)$ when the number of labels m and the feature dimensionality d are constant. While the computation of p-values involves $\mathcal{O}(n \log n)$ operations, this step is less significant than the $\mathcal{O}(n^2)$ cost of updating the distance matrix. The p-values can be computed in parallel by specifying `n_jobs` in the construction. The default is without parallelisation. At the time of writing, `online-cp` cannot handle changes in m or d over time.

For this example, the computation time is dominated by the prediction step, as we reuse the computed `D` matrix upon learning. As shown in Figure 3, the prediction and learning of the last few examples takes about 0.2 seconds. If we went on up to 2000 examples, we would be able to predict-and-learn each new example in under 0.7 seconds. Extrapolating, with 5000 examples we would expect something around 4 seconds per example and with

10000 something around 15 second per example using the same hardware. This would have a reasonable use in data that arrive every hour for a year (8760 examples). Even with theoretically bad complexity, $\mathcal{O}(n^2)$ per prediction/update, online conformal prediction can be useful in a range of applications.

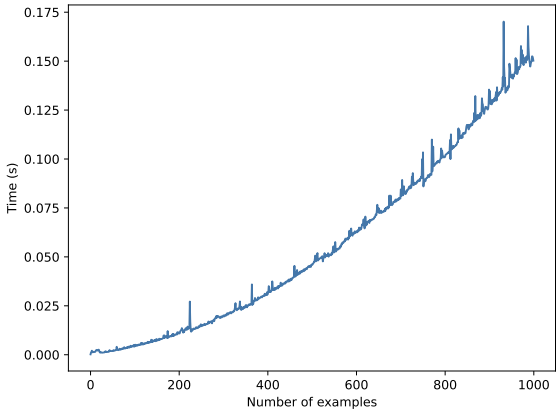


Figure 3: Prediction time per iteration for `ConformalNearestNeighboursClassifier` in our example.

4.2. Regression

We generate $N = 1000$ i.i.d. data points (X_i, Y_i) with $X_i \sim \mathcal{N}(0, I_4)$ and $Y_i \sim X_i^T \beta + \mathcal{N}(0, 1)$ for a coefficient vector $\beta = (2, 1, 0, 0)$. This synthetic dataset was processed using `ConformalRidgeRegressor`. The output of a `ConformalRegressor` is a `ConformalPredictionInterval`, which is represented as a tuple with the interval boundaries. It also has a `width` method and stores the significance level. To evaluate the regressors, we have the classes `Width` and `WinklerScore`. The former simply tracks the width of the interval, whereas the Winkler score (Winkler, 1972) is a proper scoring rule for interval forecasts (Gneiting and Raftery, 2007). It penalises intervals and adds a penalty for miscoverage, which depends on the significance level, ε. Assuming Γ^ε is an interval, the Winkler score can be written

$$S_\varepsilon^{\text{int}}(\Gamma_\varepsilon, y) = |\Gamma^\varepsilon| + \frac{2d(\Gamma^\varepsilon, y)}{\varepsilon},$$

where $d(A, \xi) = \inf\{d(\xi, a) : a \in A\}$ denotes the Euclidean distance from the point $\xi \in \mathbb{R}$ to the set $A \subset \mathbb{R}$.

The following code processes the synthetic dataset.

```
    from online_cp import ConformalRidgeRegressor
    from online_cp import WinklerScore, Width
    cp = ConformalRidgeRegressor(a=0.001, epsilon=0.1)
    metrics = Evaluation(err=Err, winkler=WinklerScore, width=Width)
    epsilon = 0.1
    # Ensure that we can get informative prediction sets
    X_init_train = X[:int(np.ceil(2/epsilon))]
    y_init_train = y[:int(np.ceil(2/epsilon))]

    X_process = X[int(np.ceil(2/epsilon)):]
    y_process = y[int(np.ceil(2/epsilon)):]

    cp.learn_initial_training_set(X_init_train, y_init_train)

    for obj, lab in zip(X_process, y_process):

        # Make prediction
        Gamma, precomputed = cp.predict(obj, return_update=True)

        # Learn the label
        cp.learn_one(obj, lab, precomputed)

        # Update efficiency criteria
        metrics.update(y=lab, Gamma=Gamma)
```

Plotting the results as in the classification setting produces the plot shown in Figure 4.

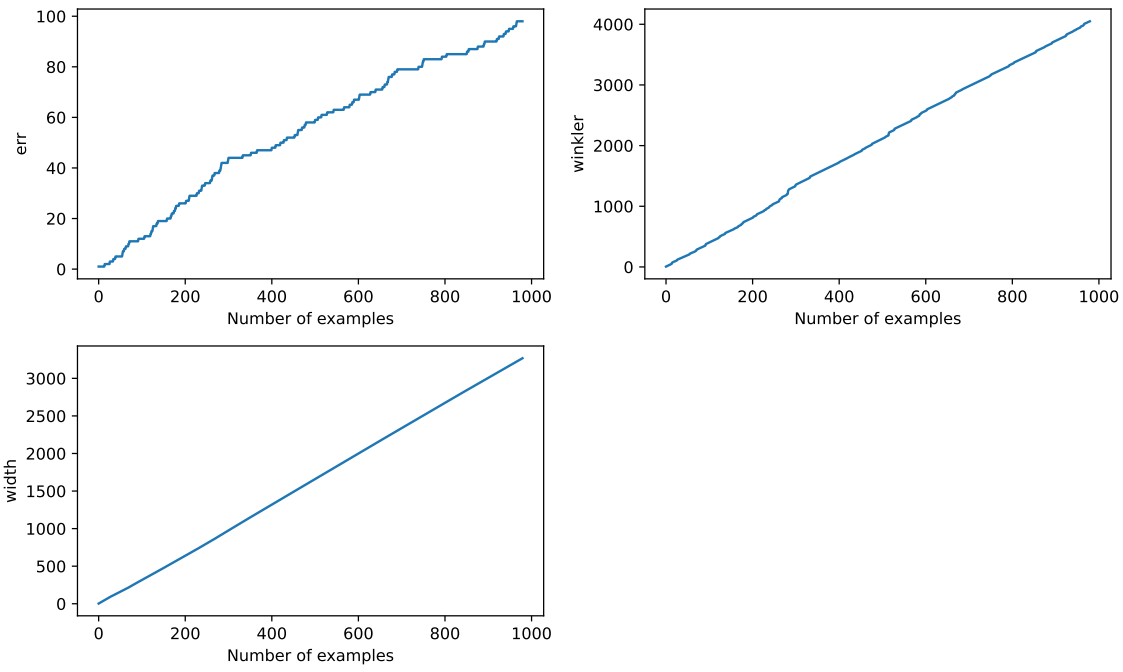


Figure 4: Cumulative results for synthetic regression.

Learning an initial training set is motivated by the fact that informative predictions require at least $\lceil 2/\varepsilon \rceil$ training examples. In general, we require at least $\lceil 1/\varepsilon \rceil$ training examples to obtain informative prediction sets because the smallest possible p-value is $1/n$ for nth example. However, the conformalized ridge regression (CRR) algorithm, which is a combination of the lower and upper CRR that produces the lower and upper bounds for the prediction interval, respectively, requires $\lceil 2/\varepsilon \rceil$ training examples. For further details, see Vovk et al. (2022). Before this point, the prediction sets are the entire real line. If `autotune` is set to true, then the ridge parameter a is tuned using generalised cross-validation (Golub et al., 1979) on the initial training set. The ridge parameter can be tuned at any point by calling `tune_ridge_parameter`. Similar to the classification task, some computations are common to `predict` and `learn_one`; therefore, we can return a precomputed object that can be used in `learn_one` to avoid repeating them. It is not necessary to invert a matrix at each step. The hat matrix can be efficiently updated online using the Sherman-Morrison formula (Sherman and Morrison, 1950), which builds on methods discussed as early as Duncan (1944).

The CRR algorithm does not explicitly compute p-values (which in principle would require an uncountable number of p-values), but if we want to compute the p-value of a particular label, the code

```
cp.compute_p_value(x=obj, y=lab, smoothed=True, precomputed=precomputed)
```

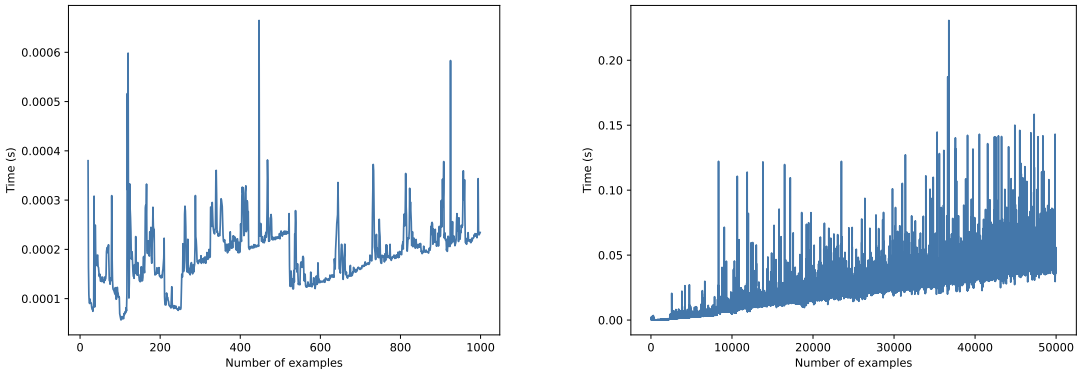
returns the smoothed p-value for the label, and setting `smoothed` to `False` returns the non-smoothed p-value.

Note that it is also possible to obtain an upper or lower bound as the prediction set. Formally, the upper prediction set is the ray $(-\infty, u)$ and is similar for the lower CRR (see Chapter 2 in Vovk et al. (2022) for the lower and upper CRR). In this case, the efficiency criteria currently implemented are not compatible because the set size is infinite.

The initial training step involves computing the inverse of the regularised kernel matrix $(X^T X + aI)^{-1}$, where X is the matrix of training samples with n rows (examples) and p columns (features, assumed constant). This step has a computational complexity of $\mathcal{O}(np^2 + p^3)$, where the $\mathcal{O}(np^2)$ term corresponds to the computation of $X^T X$ and the $\mathcal{O}(p^3)$ term corresponds to the matrix inversion. For large n, the complexity is dominated by $\mathcal{O}(np^2)$, whereas for large p, it is dominated by $\mathcal{O}(p^3)$.

Each iteration of the loop consists of a prediction step and a learning step. The prediction step involves updating the inverse of the kernel matrix using the Sherman-Morrison formula ($\mathcal{O}(1)$), computing the hat matrix ($\mathcal{O}(n^2)$), and sorting the nonconformity scores to construct the prediction interval ($\mathcal{O}(n \log n)$). However, only the diagonal, and the last column are actually needed to compute nonconformity scores, which can be done in $\mathcal{O}(n)$. After this, assuming that we return the precomputed object to avoid redundant computations, the `learn_one` is performed in time $\mathcal{O}(1)$. The dominating complexity is $\mathcal{O}(n \log n)$.

As in the classification example, the dominating computational cost is the prediction step, as shown in Figure 5(a). Extending beyond 1000 examples, we plot the prediction time per example for 50000 examples in Figure 5(b). In total, it takes about 20 minutes to process all 50000 examples. This means that processing data that arrives every ten minutes can certainly be done for at least a year (52560 examples), which indicates that online conformal prediction is feasible in a range of scenarios where new data arrive at a



(*a*) Prediction time per iteration for `ConformalRidgeRegressor` in our example.

(*b*) Prediction time per iteration for `ConformalRidgeRegressor` for 50000 examples

Figure 5: Per iteration compute time for `ConformalRidgeRegressor`.

slower rate. If we extrapolate from the graph, processing data that arrive every minute for a couple of years seems feasible too.

4.3. Conformal predictive systems

Conformal predictive systems (CPS) are discussed in detail in Vovk et al. (2022), but a theoretical discussion is beyond the scope of this paper. We just briefly mention that the output of a CPS is a conformal predictive distribution (CPD), which intuitively behaves like a predictive distribution function. We follow Vovk et al. (2022) in representing the output of the conformal predictive system Π in the form

$$\Pi_n(y) = [\Pi_n(y, 0), \Pi_n(y, 1)],$$

unless a (random) number $\tau \in [0, 1]$ is passed along with y, in which case the output is the convex combination

$$\Pi_n(y, \tau) = [(1 - \tau)\Pi_n(y, 0) + \tau\Pi_n(y, 1)], \tag{4}$$

Before processing our synthetic regression dataset, we train a CPS on the first 100 examples, and illustrate the CPD resulting from predicting the 101st example. Figure 6 shows the result of `cpd.plot()`. If we pass a random number $\tau \in [0, 1]$ to `plot`, the convex combination (4) is plotted.

```
cps = RidgePredictionMachine(a=0.001)
cps.learn_initial_training_set(X=X[:100], y=y[:100])
cpd = cps.predict_cpd(X[100])
cpd.plot()
```

The `ConformalPredictiveDistribution` class has several methods. It can be used to compute a `ConformalPredictiveInterval`

```
cpd.predict_set(tau=tau, epsilon=0.1)
```

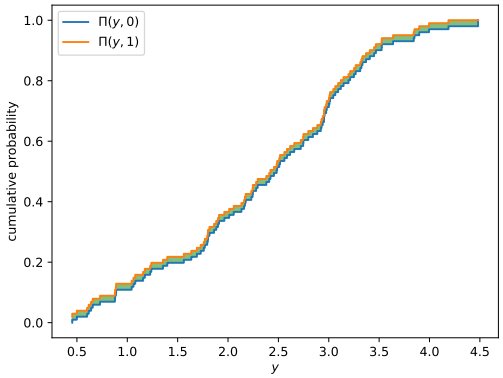


Figure 6: Plot of a conformal predictive distribution. The upper curve shows the upper distribution $\Pi_n(y, 1)$ and the lower curve shows the lower distribution $\Pi_n(y, 0)$. The green area illustrates where the curve may fall if we pass a random number τ.

but we can also optimise the width of the prediction set by

```
cpd.predict_set(tau=tau, epsilon=0.1, minimise_width=True)
```

The procedure for minimising width is described in Vilfroy et al. (2024). In the context of computing prediction intervals from a CPD, it consists of choosing two quantiles that align with the chosen significance level while minimising their distance.

There are several options for evaluation of conformal predictive systems. A simple option is to output prediction sets at a fixed significance level, minimise their width as above, and evaluate exactly as we did for the prediction intervals. We can also use the `CRPS` class, which computes the continuous ranked probability score (CRPS), which is a proper scoring rule (Gneiting and Raftery, 2007) for predictive distributions. However, we should note that the CRPS of a predictive distribution F, given the outcome y, is computed as

$$\int_{-\infty}^{\infty} (F(x) - \mathbb{1}_{\{x \geq y\}})^2 dx.$$

For a CPD Π_n, the integral becomes

$$\int_{-\infty}^{\infty} (\Pi_n(x, \tau) - \mathbb{1}_{\{x \geq y\}})^2 dx,$$

which is nonconverging because the range of $\Pi_n(x, \tau)$ is a strict subset of $[0, 1]$. The best solution we came up with, is to replace Π_n with the function

$$\overline{\Pi}_n(x) = \begin{cases} \Pi(x, 0) & \text{if } x \leq y \\ \Pi(x, 1) & \text{if } x > y, \end{cases}$$

which ensures that the CRPS converges. The following code processes the synthetic regression dataset.

```
from online_cp import RidgePredictionMachine, CRPS

cps = RidgePredictionMachine(a=0.001)

cps.learn_initial_training_set(X_init_train, y_init_train)

metrics = Evaluation(crps=CRPS)

for obj, lab in zip(X_process, y_process):
    tau = np.random.uniform(0, 1)

    # Compute CPD
    cpd, precomputed = cps.predict_cpd(x=obj, return_update=True)

    # Learn the label
    cps.learn_one(x=obj, y=lab, precomputed=precomputed)

    # Update efficiency
    metrics.update(y=lab, cpd=cpd)
```

As usual, `metrics.plot_cumulative()` produces the plot shown in Figure 7

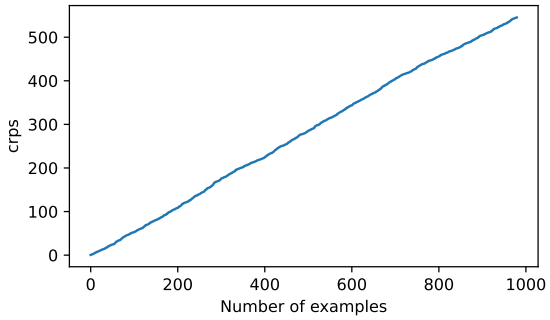


Figure 7: Cumulative results of synthetic regression with `RidgePredictionMachine`

4.4. Testing exchangeability

Currently, there are two conformal test martingales implemented in `online-cp`: the plugin martingale (Fedorova et al., 2012a) and the Simple Jumper martingale (Vovk et al., 2022). We illustrate the former in practice by modifying the synthetic regression dataset by, instead of the coefficient vector $\beta = (2, 1, 0, 0)$, using the new coefficient vector $\beta_2 = (0, 0, 1, 2)$ for the last 500 labels, thus introducing a change point at $n = 500$. In our example, the p-values were computed using a `ConformalisedRidgeRegressor`. The plugin martingale uses a kernel density estimate (KDE) (see Rosenblatt (1956) and Parzen (1962) for details) computed from the p-values observed up to the point $n-1$ for betting functions. Thus, the betting function is not static but is updated each time a new p-value is observed. When the number of observed p-values is small, the estimate is likely to be poor; to avoid losing too much capital initially, one may wish to bet cautiously, which can be achieved by mixing the

KDE with the uniform distribution on $[0, 1]$ (which avoids betting), using a mixing parameter that depends on the number of observed p-values. This can be achieved by passing the keyword `min_sample_size` (the default is 100). Initially, the betting functions are uniform, and until we reach the desired sample size, we mix in more and more of the KDE. The growth rate of the mixing parameter can be controlled by setting a `mixing_exponent`. The default is 1, which corresponds to linear growth.

For the Gaussian kernel, the bandwidth is selected automatically using Silverman's rule of thumb (Silverman, 2018). The Gaussian kernel has the disadvantage that it is not bounded; however, the p-values always lie in the unit interval. This leads to poor performance near the boundary. This is handled by reflecting the p-values to the left from 0 and to the right from 1 and fitting the KDE to the extended sequence $\{-p_i, p_i, 2-p_i\}$. The KDE-based betting strategy can also use a sliding window, fitting a betting function to only a window of temporally near p-values. This is potentially useful if there is a distribution shift, as the density estimation is less affected by the p-values from the old distribution.

The `PluginMartingale` use KDE with Gaussian kernel as the default betting strategy. However, different plugin strategies can be used. For example, the class `BetaMoments` specifies a betting strategy that estimates the parameters of a beta distribution and uses the probability density function to bet. To employ it, we pass it in the construction of the martingale.

```
from online_cp import PluginMartingale
from online_cp.martingale import BetaMoments
martignale = PluginMartingale(betting_strategy=BetaMoments())
```

Parameter estimation is then carried out online using the method of moments, or if we use the betting strategy `BetaMLE`, the parameters are estimated using the maximum likelihood method. The latter goes back to Fisher (1922), while the former was used by Chebyshev in his proof of the central limit theorem, and popularised by Pearson as a practical method for parameter estimation. The `BetaMoments` strategy has the advantage of being much faster to compute than KDE, and the beta distribution is flexible enough to bet on many deviations from uniformity. Even custom betting strategies can be passed to the constructor, which makes the `PluginMargingale` a flexible tool for testing exchangeability.

When constructing a `ConformalTestMartingale`, we can specify a warning level corresponding to C in (3). If the martingale reaches the warning level, a user warning is raised, indicating that the exchangeability assumption is likely violated. The default plugin martingale uses the Gaussian kernel.

We process the synthetic regression dataset with a change point as follows:

```
S1 = GaussianKDE()
S2 = BetaMoments()
S3 = BetaMLE()
M1 = PluginMartingale(betting_strategy=S1, warnings=False)
M2 = PluginMartingale(betting_strategy=S2, warnings=False)
M3 = PluginMartingale(betting_strategy=S3, warnings=False)
M4 = SimpleJumper(warnings=False)

cp = ConformalRidgeRegressor(a=0.001)

for obj, lab in zip(X, y):
```

```
12
13          # Learn the label
14          cp.learn_one(obj, lab)
15
16          # Compute the smoothed p-value
17          p = cp.compute_p_value(x=obj, y=lab, smoothed=True)
18
19          # Update martingales
20          M1.update_martingale_value(p)
21          M2.update_martingale_value(p)
22          M3.update_martingale_value(p)
23          M4.update_martingale_value(p)
```

The resulting martingale values are shown in Figure 8.

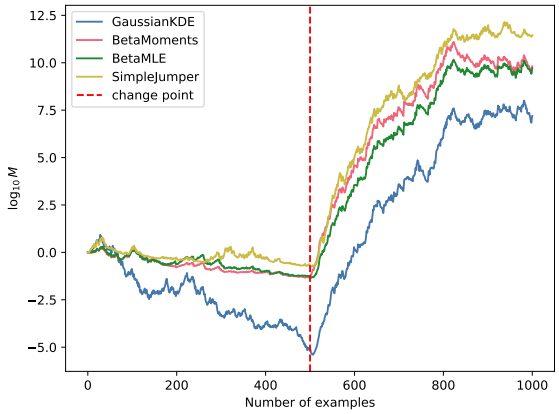


Figure 8: The plugin martingale with `GaussianKDE`, `BetaMoments`, `BetaMLE`, and `SimpleJumper` for the synthetic regression dataset with a change point at $n = 500$. The p-values are generated with the CRR algorithm.

5. Concluding remarks and future work

We have presented some examples that illustrate the `online-cp` package in practice for conformal prediction in the online setting. We hope that this contribution to the growing collection of open-source software for conformal prediction can help make it convenient for researchers and practitioners to use conformal prediction in its natural habitat, the online setting, where the validity guarantees hold in their strongest form.

The package is still in development, and several methods have yet to be implemented. In its current form, the kernel method for conformal regression and conformal predictive systems is implemented, but should be considered experimental, as well as the Nearest Neighbours Prediction Machine, a conformal predictive system, based on the nearest neighbours algorithm. Future plans include the conformalized Lasso and elastic net algorithms (Lei, 2019), support vector machines (Cortes and Vapnik, 1995), and more test martingales. Another key direction for future work is the implementation of Mondrian conformal

predictors (Vovk et al., 2022). Moving beyond the exchangeability model, other online compression models such as the Gauss Linear model (Fedorova et al., 2012b) and hypergraphical models (Fedorova et al., 2015) may be considered in future work.

Conformal prediction in the online setting is notoriously computationally heavy because it requires frequent retraining of the underlying machine learning models. An interesting research direction may be to conformalize dedicated online learning algorithms. However, a major challenge for such a program is that a (non)conformity measure may not depend on the order in which examples are observed, excluding many existing online learning methods. However, our runtime measurements showed that online conformal prediction may be feasible when examples arrive at a slow rate. If examples arrive at a scale of several minutes to an hour, it may be feasible to run the model on a yearly scale.

Finally, we extend an open invitation to anyone who may be interested in contributing to the development of `online-cp` and encourage users to report any bugs or suggestions on improvements that may arise.

Acknowledgments

The authors acknowledge the Swedish Knowledge Foundation and industrial partners for financially supporting the research and education environment on Knowledge Intensive Product Realisation SPARK at Jönköping University, Sweden. Project: PREMACOP grant no. 20220187.

References

Patrick Altmeyer and contributors. Conformalprediction.jl: Predictive uncertainty quantification through conformal prediction for machine learning models trained in mlj. https://github.com/JuliaTrustworthyAI/ConformalPrediction.jl, 2023. Version 0.1.1.

Henrik Boström. crepes: a python package for generating conformal regressors and predictive systems. In Ulf Johansson, Henrik Boström, Khuong An Nguyen, Zhiyuan Luo, and Lars Carlsson, editors, *Proceedings of the Eleventh Symposium on Conformal and Probabilistic Prediction and Applications*, volume 179 of *Proceedings of Machine Learning Research*. PMLR, 2022.

Thibault Cordier, Vincent Blot, Louis Lacombe, Thomas Morzadec, Arnaud Capitaine, and Nicolas Brunel. Flexible and Systematic Uncertainty Estimation with Conformal Prediction via the MAPIE library. In *Conformal and Probabilistic Prediction with Applications*, 2023.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20: 273–297, 1995.

W.J. Duncan. Lxxviii. some devices for the solution of large sets of simultaneous linear equations. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 35(249):660–670, 1944. doi: 10.1080/14786444408520897. URL https://doi.org/10.1080/14786444408520897.

Valentina Fedorova, Alex Gammerman, Ilia Nouretdinov, and Vladimir Vovk. Plug-in martingales for testing exchangeability on-line. In John Langford and Joelle Pineau, editors, *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 1639–1646. Omnipress, 2012a. ISBN 978-1-4503-1285-1.

Valentina Fedorova, Ilia Nouretdinov, and Alex Gammerman. Testing the gauss linear assumption for on-line predictions. *Progress in Artificial Intelligence*, 1:205–213, 2012b.

Valentina Fedorova, Alex Gammerman, Ilia Nouretdinov, and Vladimir Vovk. Hypergraphical conformal predictors. *International Journal on Artificial Intelligence Tools*, 24(06): 1560003, 2015.

Ronald A Fisher. On the mathematical foundations of theoretical statistics. *Philosophical transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character*, 222(594-604):309–368, 1922.

Niharika Gauraha and Ola Spjuth. conformalclassification: A conformal prediction r package for classification. *arXiv preprint arXiv:1804.05494*, 2018.

Tilmann Gneiting and Adrian E Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American statistical Association*, 102(477):359–378, 2007.

Gene H Golub, Michael Heath, and Grace Wahba. Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics*, 21(2):215–223, 1979.

Jing Lei. Fast exact conformalization of the lasso using piecewise linear homotopy. *Biometrika*, 106(4):749–764, 2019.

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

Jacob Montiel, Max Halford, Saulo Martiello Mastelini, Geoffrey Bolmier, Raphael Sourty, Robin Vaysse, Adil Zouitine, Heitor Murilo Gomes, Jesse Read, Talel Abdessalem, et al. River: machine learning for streaming data in python. 2021.

Emanuel Parzen. On Estimation of a Probability Density Function and Mode. *The Annals of Mathematical Statistics*, 33(3):1065 – 1076, 1962. doi: 10.1214/aoms/1177704472. URL https://doi.org/10.1214/aoms/1177704472.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Murray Rosenblatt. Remarks on Some Nonparametric Estimates of a Density Function. *The Annals of Mathematical Statistics*, 27(3):832 – 837, 1956. doi: 10.1214/aoms/1177728190. URL https://doi.org/10.1214/aoms/1177728190.

Jack Sherman and Winifred J Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, 21(1): 124–127, 1950.

Bernard W Silverman. *Density estimation for statistics and data analysis*. Routledge, 2018.

Solène Vilfroy, Lionel Bombrun, Thierry Urruty, Florence De Grancey, Jean-Philippe Lebrat, and Philippe Carré. Conformal prediction for regression models with asymmetrically distributed errors: application to aircraft navigation during landing maneuver. *Machine Learning*, 113(10):7841–7866, 2024.

Jean Ville. *Etude critique de la notion de collectif*. Gauthier-Villars Paris, 1939.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Vladimir Vovk, Ilia Nouretdinov, and Alexander Gammerman. Testing exchangeability online. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pages 768–775, 2003.

Vladimir Vovk, Alexander Gammerman, and Glenn Shafer. *Algorithmic learning in a random world*, volume 29. Springer, 2005.

Vladimir Vovk, Alexander Gammerman, and Glenn Shafer. *Algorithmic Learning in a Random World*. Springer, second edition, December 2022. ISBN 978-3-031-06648-1. doi: https://doi.org/10.10007/978-3-031-06649-8.

Robert L Winkler. A decision-theoretic approach to interval estimation. *Journal of the American Statistical Association*, 67(337):187–191, 1972.