libconform v0.1.0: a Python library for conformal prediction

Jonas Faßbender JONAS@FASSBENDER.DEV

1. Introduction

This paper introduces the Python library `libconform`, implementing concepts defined in Vovk et al. (2005), namely the conformal prediction framework and Venn prediction for reliable machine learning. These algorithms address a weakness of more traditional machine learning algorithms which produce only bare predictions, without their confidence in them/the probability of the prediction, therefore providing no measure of likelihood, desirable and even necessary in many real-world application domains.

The conformal prediction framework is composed of variations of the conformal prediction algorithm (CP), first described in Vovk et al. (1999); Saunders et al. (1999). A conformal predictor provides a measurement of confidence in its predictions. A Venn predictor, on the other hand, provides a multi-probabilistic measurement, making it a probabilistic predictor. Below in the text, Venn predictors are included if only "conformal prediction framework" is written, except stated otherwise.

The conformal prediction framework is applied successfully in many real-world domains, for example face recognition, medical diagnostic and prognostic and network traffic classification (see Balasubramanian et al., 2014, Part 3).

It is build on traditional machine learning algorithms, the so called underlying algorithms (see Papadopoulos et al., 2007), which makes Python the first choice for implementation, since its machine learning libraries are top of the class, still evolving and improving due to the commitment of a great community of developers and researchers.

`libconform`'s aim is to provide an easy to use, but very extensible API for the conformal prediction framework, so developers can use their preferred implementations for the underlying algorithm and can leverage the library, even in this early stage. `libconform v0.1.0` is **not** yet stable; there are still features missing and the API is very likely to change and improve. The library is licensed under the MIT-license and its source code can be downloaded from `https://github.com/jofas/conform`.

This paper combines `libconform`'s documentation with a outline of the implemented algorithms. At the end of each chapter there are notes on the implementation containing general information about the library, descriptions of the internal workings and the API and possible changes in future versions.

Appendix A provides an overview over `libconform`'s API and Appendix B contains examples on how to use the library.

2. Conformal predictors

Like stated in the introduction, this chapter will only outline conformal prediction (CP). For more details see Vovk et al. (2005).

CP—like the name suggests—determines the label(s) of an incoming observation based on how well it/they conform(s) with previous observed examples. Conformal prediction can be used either in the online or the offline, or batch learning setting. The offline learning setting, compared to online learning, weakens the validity—described below in this chapter—of the classifier in favor of computational efficiency (see Vovk et al., 2005, Chapter 4).

Let $\wr z_1, \ldots, z_n \wr$ be a bag, also called multiset[1], of examples, where each example $z_i \in \mathbf{Z}$ is a tuple $(x_i, y_i); x_i \in \mathbf{X}, y_i \in \mathbf{Y}$. $\mathbf{X}$ is called the observation space and $\mathbf{Y}$ the label space. For this time $\mathbf{Y}$ is considered finite, making the task of prediction a classification task, rather than regression, which will be considered in Chapter 2.2.

A conformal predictor can be defined as a confidence predictor Γ. For this an input $\epsilon \in (0, 1)$, the significance level is needed. $1 - \epsilon$ is called the confidence level. A conformal predictor Γ^ϵ in the online setting is conservatively valid under the exchangeability assumption, which means, as long as exchangeability holds, it makes errors at a frequency of ϵ or less. For more on that refer to Vovk et al. (2005, Chapters 1–4,7).

CP, in its original setting, produces nested prediction sets. Rather than returning a single label as its prediction, it returns a set of elements $\mathbf{Y'} \in 2^\mathbf{Y}$, $2^\mathbf{Y}$ being the set of all subsets of $\mathbf{Y}$, including the empty set. The prediction sets are called nested, because, for $\epsilon_1 \geq \epsilon_2$, the prediction of Γ^{ϵ_1} is a subset of Γ^{ϵ_2} (see Vovk et al., 2005, Chapter 2).

In order to predict the label of a new observation x_{n+1}, set $z_{n+1} := (x_{n+1}, y)$, for each $y \in \mathbf{Y}$ and check how z_{n+1} conforms with the examples of our bag $\wr z_1, \ldots, z_n \wr$.

This is done with a nonconformity measure $A_{n+1} : \mathbf{Z}^n \times \mathbf{Z} \to \mathbb{R}$. First, z_{n+1} is added to the bag, then A_{n+1} assigns a numerical score to each example in z_i:

$$\alpha_i = A_{n+1}(\wr z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, z_{n+1} \wr, z_i). \tag{1}$$

One can see in this equation that z_i is removed from the bag. It is also possible to compute α_i with z_i in the bag, which means for $A_{n+1} : \mathbf{Z}^{n+1} \times \mathbf{Z} \to \mathbb{R}$ the score is computed as:

$$\alpha_i = A_{n+1}(\wr z_1, \ldots, z_{n+1} \wr, z_i). \tag{2}$$

Which one is preferable is case-dependent (see Shafer and Vovk, 2008, Chapter 4.2.2).

1. It is typical in machine learning to denote this as a data set, even though examples do not have to be unique, making the so called set a multiset. A multiset is not a list, since the ordering of the elements is not important.

α_i is called nonconformity score. The nonconformity score can now be used to compute the p-value for z_{n+1}, which is the fraction of examples from the bag which are at least as nonconforming as z_{n+1}:

$$\frac{|\{i = 1, \ldots, n+1 : \alpha_i \geq \alpha_{n+1}\}|}{n+1}. \tag{3}$$

Another way to determine the p-value is through smoothing, in which case the nonconformity scores equal to α_{n+1} are multiplied by a random value τ_{n+1}:

$$\frac{|\{i = 1, \ldots, n+1 : \alpha_i > \alpha_{n+1}\}| + \tau_{n+1}|\{i = 1, \ldots, n+1 : \alpha_i = \alpha_{n+1}\}|}{n+1}. \tag{4}$$

A conformal predictor using the smoothed p-value is called a smoothed conformal predictor and is exactly valid under exchangeability in the online setting, which means it makes errors at a rate exactly ϵ (see Vovk et al., 2005, Chapter 2). If the p-value of z_{n+1} is larger than ϵ, y is added to the prediction set.

Algorithm 1 : Conformal predictor $\Gamma^{\epsilon}(\wr z_1, \ldots, z_n \wr, x_{n+1})$

for all $y \in \mathbf{Y}$ **do**
  set $z_{n+1} := (x_{n+1}, y)$ and add it to the bag
  for all $i = 1, \ldots, n+1$ **do**
    compute α_i with (1) or (2)
  end for
  set p_y with (3) or (4)
  if $p_y > \epsilon$ **then**
    add p_y to prediction set
  end if
end for
return prediction set

Notes on the implementation: `libconform` provides the `CP` class for creating conformal prediction classifiers. `libconform`'s classifier classes provide quite equal APIs, only with minor variations. The API of the classifier classes is comparable to major machine learning libraries like sklearn or keras (see Buitinck et al., 2013; Chollet et al., 2015).

It is common in machine learning to split the learning task in two distinct operations, first a training—or fit—operation on a training set of examples and then a predict operation on new observations. `libconform`'s classifier classes follow this style, providing a `train` and a `predict` method.

While this split in training and predicting is common for inductive classifiers, which first derive a prediction rule, or decision surface, from the training set and then predict

unseen examples inductively based on that rule, it is not really the way CP works. CP was designed to be transductive, not inductive, which means rather than to generate a prediction rule, it uses all previous seen examples to classify a new observation, making the training step unnecessary (see Algorithm 1). While the transductive setting is more elegant than the inductive setting, it is computationally very expensive and not feasible for larger data sets and for underlying algorithms—discussed in Chapter 2.1—which have a computationally complex training phase (see Papadopoulos et al., 2007; Vovk et al., 2005, Chapter 1).

`libconform`'s aim is to be—one day—ready for production, where, for some application domains, the time complexity of predicting a new observation is crucial, while the time complexity of the training phase is—in a certain range—not as important. Therefore `libconform`'s `CP` class tries to minimize the time complexity of its `predict` method. Instead of adding z_{n+1} to the bag and then computing α_i for each example in the bag during prediction, it computes $\alpha_1, \ldots, \alpha_n$ during training and only computes α_{n+1} in `predict` (see Algorithm 1, lines 3–5). Therefore—not adding z_{n+1} to the bag—it currently computes the nonconformity scores based on A_n instead of A_{n+1}.

Arguably `CP` does not implement the conformal prediction algorithm in its original form, being currently rather a special case of inductive conformal prediction, where the calibration set is equal to the whole bag of examples previously witnessed, instead of a subset (see Chapter 3). It is possible that `CP` will change to being the implementation of the original conformal prediction algorithm described in Vovk et al. (2005, Chapter 2) in a future version, or simply providing an extra method for the computationally more demanding original online learning setting defined in Vovk et al. (2005, Chapter 2).

`CP` takes an instance of a nonconformity measure A and a sequence of $\epsilon_1, \ldots, \epsilon_g$ as its arguments during initialization, therefore being the implementation of $\Gamma^{\epsilon_1}, \ldots, \Gamma^{\epsilon_g}$.

It also provides to extra utility methods for validation, `score` and `score_online`, which generate metrics for the conformal predictors $\Gamma^{\epsilon_1}, \ldots, \Gamma^{\epsilon_g}$. The most important of those metrics are the error rates $Err_1, \ldots, Err_g$. It the error rate $Err_i \leq \epsilon_i$ over the bag of examples provided to `score`/`score_online` than Γ^{ϵ_i} was valid on the bag.

`score_online` adds an example, after it was predicted, to the training bag and calls `train`, using $\Gamma^{\epsilon_1}, \ldots, \Gamma^{\epsilon_g}$ in the online learning setting.

2.1 Nonconformity measures based on underlying algorithms

Previously nonconformity measures were only described as any function $A : \mathbf{Z}^* \times \mathbf{Z} \to \mathbb{R}$, $\mathbf{Z}^*$ being any possible bag of examples from $\mathbf{Z}$. This chapter will make a more concrete description on what nonconformity measures are and how they use underlying traditional machine learning algorithms.

Let $D : \mathbf{Z}^* \times \mathbf{X} \to \hat{\mathbf{Y}}$ be a traditional machine learning algorithm. $\hat{\mathbf{Y}}$ must not be equal to $\mathbf{Y}$. Furthermore there exists a discrepancy measure $\Delta : \mathbf{Y} \times \hat{\mathbf{Y}} \to \mathbb{R}$. For a concrete bag $\wr z_1, \ldots, z_n \wr$, $D_{\wr z_1, \ldots, z_n \wr}$ would be the instance of D trained on the bag, generating a

decision surface based on it. $D_{(\!|z_1,\ldots,z_n|\!)}(x)$ would return the label $\hat{y}$ for x. Now we can define the nonconformity score α for $z := (x, y)$ from the nonconformity measure A_n as:

$$\alpha = A_n((\!|z_1, \ldots, z_n|\!), z) = \Delta\big(y, D_{(\!|z_1,\ldots,z_n|\!)}(x)\big),$$

or rather with removed example for any α_i, $i = 1, \ldots, n$:

$$\alpha_i = A_n((\!|z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, z_n|\!), z_i) = \Delta\big(y, D_{(\!|z_1,\ldots,z_{i-1},z_{i+1},\ldots,z_n|\!)}(x_i)\big).$$

Especially the second equation can be computationally very complex since it requires to refit D for each $i = 1, \ldots, n$. In general it is not very natural to use a inductive decision surface D within the transductive framework of CP.

A popular nonconformity measure is based on the nearest neighbor method (see Vovk et al., 2005; Shafer and Vovk, 2008; Balasubramanian et al., 2014; Smirnov et al., 2009). The general description for the k-nearest neighbor method can be found in Smirnov et al. (2009), the other articles/books describe the nonconformity measure based on the 1-nearest neighbor method for $z := (x, y)$ as:

$$A_n((\!|z_1, \ldots, z_n|\!), z) = \frac{\min_{i=1,\ldots,n:y_i=y} d(x, x_i)}{\min_{i=1,\ldots,n:y_i\neq y} d(x, x_i)},$$

d being a distance measure, for example the Euclidean distance. It should be noted that A_n based on the 1-nearest neighbor method for $A_n((\!|z_1, \ldots, z_n|\!), z_i), i = 1, \ldots, n$ requires the removal of z_i from the bag, since otherwise the smallest distance for $y_i = y_j, j = 1, \ldots, n$ would always be 0 resulting in worthless nonconformity scores.

The more general nonconformity measure based on the k-nearest neighbor method can be described as:

$$A_n((\!|z_1, \ldots, z_n|\!), z) = \frac{d_k^y}{d_k^{-y}},$$

d_k being the sum of the k smallest distances to x, $-y$ being all the examples where $y \neq y_i, i = 1, \ldots, n$.

Notes on the implementation: `libconform` tries again to be as extensible as possible, providing a way for developers to define their own nonconformity measures. For nonconformity measures `libconform` provides a module `ncs` containing predefined nonconformity measures and a base class for inheritance called `NCSBase`.

Predefined are currently the k-nearest neighbor method, one based on a decision tree (see Vovk et al., 2005, Chapter 4) and one based on neural networks (see Papadopoulos et al., 2007; Vovk et al., 2005, Chapter 4). The k-nearest neighbor method and the decision tree are based on the sklearn library (see Buitinck et al., 2013).

Nonconformity measures are classes inheriting from `NCSBase` and have to provide an interface with three methods: `train`, `scores` and `score`.

`train`: $\mathbf{X}^n \times \mathbf{Y}^n$ is for fitting the underlying algorithm D to a bag of examples.

`scores`: $\mathbf{X}^m \times \mathbf{Y}^m \times bool \to \mathbb{R}^m$ returning the scores for a bag of examples. The *bool* value provided as a parameter tells the nonconformity measure if the bag is equal to the bag provided to `train`, making it possible to implement (1), rather than (2). The CP-implementation passes the same bag to `train` and `scores`, while the inductive conformal prediction implementation (see Chapter 3) passes another bag—the so called calibration set—as a parameter to `scores`.

`score`: $\mathbf{X} \times \mathbf{Y}^{|\mathbf{Y}|} \to \mathbb{R}^{|\mathbf{Y}|}$ is for returning the scores of an example x and each $y \in \mathbf{Y}$ combined as $z := (x, y)$.

2.2 Conformal predictor for regression: ridge regression confidence machine

The ridge regression confidence machine algorithm is described in Nouretdinov et al. (2001); Vovk et al. (2005, Chapter 2.3). In this chapter, unlike in the previous and following chapters, $\mathbf{Y}$ will be $\mathbb{R}$, making the prediction a regression problem instead of classification.

Algorithm 1 is not feasible for regression, since $\mathbf{Y}$ is now infinite and we would need to test for each $y \in \mathbf{Y}$ if it is in the prediction set or not. Instead the ridge regression confidence machine (RRCM) algorithm offers a different approach, returning prediction intervals instead of prediction sets.

Even though RRCM has ridge regression in its name, it can be used with other underlying algorithms, like nearest neighbor regression. For more on ridge regression and its special case linear regression refer to e.g. Hastie et al. (2009, Chapter 3).

Let $\wr z_1, \ldots, z_n \wr$ be our bag of examples, let $z_{n+1} := (x_{n+1}, y)$ be the observation we want to predict and let $D_{\wr z_1,\ldots,z_{n+1} \wr}$ be an underlying regression algorithm. Previously nonconformity scores were treated as constants, now we treat them as functions, since y is now an unknown variable: $\alpha_i(y) = |a_i + b_i y|, i = 1, \ldots, n+1$. a_i and b_i are provided by the underlying regression algorithm. Each b_i is always positive, if not a_i and b_i are multiplied with -1.

Now we can compute the set of y's which p-values are exceeding a significance level ϵ. Let $S_i = \{y : |a_i + b_i y| \geq |a_{n+1} + b_{n+1} y|\}, i = 1, \ldots, n$. Each S_i looks like:

$$S_i = \begin{cases} [u_i, v_i] & \text{if } b_{n+1} > b_i \\ (-\infty, u_i] \cup [v_i, \infty) & \text{if } b_{n+1} < b_i \\ [u_i, \infty) & \text{if } b_{n+1} = b_i > 0 \text{ and } a_{n+1} < a_i \\ (-\infty, u_i] & \text{if } b_{n+1} = b_i > 0 \text{ and } a_{n+1} > a_i \\ \mathbb{R} & \text{if } b_{n+1} = b_i = 0 \text{ and } |a_{n+1}| \leq |a_i| \\ \emptyset & \text{if } b_{n+1} = b_i = 0 \text{ and } |a_{n+1}| > |a_i| \end{cases},$$

so each S_i is either an interval, a point (a special interval), the union of two rays, a ray, the real line or empty. u_i and v_i are either the minimum/maximum of $-\frac{a_i - a_{n+1}}{b_i - b_{n+1}}$ and $-\frac{a_i + a_{n+1}}{b_i + b_{n+1}}$, if $b_{n+1} \neq b_i$ or $u_i = v_i = -\frac{a_i + a_n}{2b_i}$, if $b_{n+1} = b_i > 0$ The p-value can only change at u_i's or v_i's. Therefore all u_i's and v_i's are sorted in ascending order generating the sequence $s_1, \ldots, s_m$ plus two more s-values, $s_0 = -\infty$, $s_{m+1} = \infty$. The p-value is constant on any interval $(s_i, s_{i+1}), i = 0, \ldots, m$ from the sorted set (see Nouretdinov et al., 2001).

After that N and M are computed from the sequence. $N_j, j = 0, \ldots, m$ for the interval (s_j, s_{j+1}) is the count of $S_i : (s_j, s_{j+1}) \subseteq S_i, i = 1, \ldots, n$. $M_j, j = 1, \ldots, m$, on the other hand does the same count only for single s_j's: $S_i : s_j \in S_i, i = 1, \ldots, n$.

For a given significance level ϵ the prediction interval is the union of intervals from N and points from M for which $\frac{N_j}{n+1} > \epsilon$ or $\frac{M_j}{n+1} > \epsilon$, respectively (see Vovk et al., 2005, Chapter 2.3).

Notes on the implementation: The ridge regression confidence machine is implemented as a class `RRCM`. It provides the same API as `CP`. It implements a computationally less complex prediction method than the one described above. While the RRCM described above runs at $\mathcal{O}(n^2)$, `RRCM` takes only $\mathcal{O}(n \log n)$, because it does not compute N and M directly but instead

$$N'_j = \begin{cases} N_j - N_{j-1} & \text{if } j = 0, \ldots, m \\ 0 & \text{if } j = -1 \end{cases}$$

and

$$M'_j = \begin{cases} M_j - M_{j-1} & \text{if } j = 1, \ldots, m \\ 0 & \text{if } j = 0 \end{cases},$$

which takes only $\mathcal{O}(n)$, making the sorting the u_i's and v_i's the most complex task (see Vovk et al., 2005, Chapter 2.3).

`RRCM` is based on underlying regression algorithms providing it with its a_i's and b_i's. Currently the library provides one of these regression algorithms, based on the k-nearest neighbor method. For $b_i, i = 1, \ldots, n$ it returns 0, for b_{n+1} it returns 1. a_i is the difference between y_i and the average of the labels of its k-nearest neighbors. y_{n+1} is set to 0 therefore the k-nearest neighbor method returns the negated average of the labels of the k-nearest neighbors of x_{n+1} as a_{n+1}.

For developing underlying regression scorers there exists a base class for inheritance called `NCSBaseRegressor`. It provides a comparable API to `NCSBase`—the base class for nonconformity measures—described in the previous chapter. Like `NCSBase` the API contains a `train` method for training the underlying algorithm. Instead of `scores` and `score` it has `coeffs` and `coeffs_n`. The first returns for a bag two vectors of coefficients a_i and

b_i for each element in the bag. `coeffs_n` returns the coefficients for the observation that needs to be predicted, in this chapter $z_{n+1} := (x_{n+1}, y)$.

3. Inductive conformal predictors

4. Mondrian (inductive) conformal predictors

5. Probabilistic prediction: Venn predictors

6. Meta-conformal predictors

7. Conclusion

Appendices

A. API reference

B. Examples

References

Vineeth Balasubramanian, Shen-Shyang Ho, and Vladimir Vovk. *Conformal Prediction for Reliable Machine Learning: Theory, Adaptations and Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, first edition, 2014. ISBN 0123985374, 9780123985378.

Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013.

François Chollet et al. Keras. `https://keras.io`, 2015.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, second edition, 2009.

Ilia Nouretdinov, Tom Melluish, and Vladimir Vovk. Ridge regression confidence machine. In *In Proceedings of the Eighteenth International Conference on Machine Learning*, pages 385–392. Morgan Kaufmann, 2001.

Harris Papadopoulos, Vladimir Vovk, and Alex Gammerman. Conformal prediction with neural networks. volume 2, pages 388 – 395, 11 2007. ISBN 978-0-7695-3015-4. doi: 10.1109/ICTAI.2007.47.

Craig Saunders, Alex Gammerman, and Vladimir Vovk. Transduction with confidence and credibility. 1999.

Glenn Shafer and Vladimir Vovk. A tutorial on conformal prediction. *Journal of Machine Learning Research*, 9:371–421, jun 2008. ISSN 1532-4435.

Evgueni Smirnov, Georgi Nalbantovi, and A. M. Kaptein. Meta-conformity approach to reliable classification. *Intelligent Data Analysis*, 13, 01 2009. doi: 10.3233/IDA-2009-0400.

Vladimir Vovk, Alex Gammerman, and Craig Saunders. Machine-learning applications of algorithmic randomness. 1999.

Vladimir Vovk, Alex Gammerman, and Glenn Shafer. *Algorithmic Learning in a Random World*. Springer, 2005.