Exact Optimization of Conformal Predictors via Incremental and Decremental Learning

Giovanni Cherubin [1] **Konstantinos Chatzikokolakis** [2] **Martin Jaggi** [3]

Abstract

Conformal Predictors (CP) are wrappers around ML models, providing error guarantees under weak assumptions on the data distribution. They are suitable for a wide range of problems, from classification and regression to anomaly detection. Unfortunately, their very high computational complexity limits their applicability to large datasets. In this work, we show that it is possible to speed up a CP classifier considerably, by studying it in conjunction with the underlying ML method, and by exploiting incremental&decremental learning. For methods such as k-NN, KDE, and kernel LS-SVM, our approach reduces the running time by one order of magnitude, whilst producing exact solutions. With similar ideas, we also achieve a linear speed up for the harder case of bootstrapping. Finally, we extend these techniques to improve upon an optimization of k-NN CP for regression. We evaluate our findings empirically, and discuss when methods are suitable for CP optimization.

1 Introduction

Conformal prediction refers to a set of techniques providing error guarantees on the predictions of an ML algorithm (Vovk et al., 2005). Its increasing popularity is due to the fact that these guarantees do not require strict assumptions on the underlying data distribution; one only needs to assume that the observed examples are exchangeable (i.e., any permutation of them is equally likely to appear) – a weaker requirement than IID. These guarantees hold for any desired ML algorithm, even if underspecified or overfitting.

A conformal predictor (CP) can be instantiated for various tasks: classification and regression (Vovk et al., 2005), anomaly detection (Laxhammar & Falkman, 2010), and clustering (Cherubin et al., 2015). Furthermore, they can be used to test if data is exchangeable (or IID) (Vovk et al., 2003). Our work focuses on classification, and it can be directly applied to tasks such as anomaly detection, clustering, and sequence prediction (Section 9). We discuss CP regression separately, in Section 8.

In this paper, we consider the original definition of CP (also referred to as "full" or transductive CP), which is known to have a good predictive power and to attain the desired coverage intervals. Unfortunately, full CP requires running a leave-one-out (LOO) procedure on the entire training set for every test point. This makes its complexity prohibitive for most real world cases: if training the ML method on n examples takes $T(n)$ time, the cost of a CP prediction for m test points is proportional to $\mathcal{O}(T(n)n\ell m)$, for a training set of n examples in an ℓ-label classification setting. A number of time-efficient modifications of CP exist (Vovk et al., 2005; Vovk, 2015; Carlsson et al., 2014; Barber et al., 2019), which although have a weaker predictive power and/or coverage guarantees (e.g., Linusson et al. (2014)).

In this work, we focus on exact optimizations of the full CP classification algorithm. We first observe that, while a CP can be constructed around virtually any ML method, most applications of CP classification only use a handful of models. It therefore makes sense to optimize the CP routine in conjunction with its underlying model. In this paper, we use this idea and exploit incremental&decremental learning principles to produce *exact optimizations* of CP for: i) k-NN, ii) Kernel Density Estimation (KDE), iii) kernel Least-Squares SVM (LS-SVM); all these reduce the complexity by at least one order of magnitude. Furthermore, iv) we show that bootstrapping methods can be marginally improved by similar ideas, and v) we extend our optimizations to CP regression. Our results demonstrate that full CP is practical for several choices of underlying methods.

1.1 Related work

We first review computationally-efficient alternatives to CP, and then discuss related work on full CP optimization.

Alternatives to full CP. Despite the desirable properties of full CP, its computational complexity makes it impractical

[1]Alan Turing Institute, London, UK [2]University of Athens [3]EPFL. Correspondence to: Giovanni Cherubin <gcherubin@turing.ac.uk>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

Table 1. Time complexity of the optimized (our contribution) and standard nonconformity measures used for full CP classification. Complexities refer to an ℓ-label classification setting, with n training and m test examples. Standard full CP requires no training.

Full CP		Train	Predict	Exact optimization
(Simplified) k-NN	Standard	$\mathcal{O}(1)$	$\mathcal{O}(n^2\ell m)$	
	Optimized	$\mathcal{O}(n^2)$	$\mathcal{O}(n\ell m)$	✓
KDE	Standard	$\mathcal{O}(1)$	$\mathcal{O}(P_K n^2 \ell m)$	
	Optimized	$\mathcal{O}(P_K n^2)$	$\mathcal{O}(P_K n\ell m)$	✓
	P_K: time complexity of computing kernel K for 1 point			
LS-SVM	Standard	$\mathcal{O}(1)$	$\mathcal{O}(n^{\omega+1}\ell m)$	
	Optimized	$\mathcal{O}(n^\omega)$	$\mathcal{O}(q^3 n\ell m)$	✓
	q: dimensionality of feature vector $\phi(x)$. For $\omega \in [2,3]$, n^ω is the training cost of an LS-SVM model.			
Bootstrap	Standard	$\mathcal{O}(1)$	$\mathcal{O}(S_g(n)Bn\ell m)$	
	Optimized	$\mathcal{O}(S_g(n)e^{-1}Bn)$	$\mathcal{O}(S_g(n)(1-e^{-1})Bn\ell m)$	✗
	B: n. classifiers			
	$S_g(n) = T_g(n) + P_g(1)$: time to train base classifier on n examples and make one prediction			

for most applications. Researchers have therefore been investigating modifications of CP, to reduce the computational complexity. For example, Inductive CP (ICP), also referred to as "split CP", trains the underlying ML method only on part of the training set, which enables it to avoid the costly LOO procedure of full CP; however, this has an impact on its prediction power (e.g., Appendix G). Several methods were proposed after ICP, such as cross-CP (Vovk, 2015), aggregated CP (Carlsson et al., 2014), CV+ and the jackknife+ (Barber et al., 2019). These methods mitigated ICP's statistical inefficiency, whilst preserving a good computational complexity. However, they have a weaker prediction power than the full CP formulation (Linusson et al., 2014; Carlsson et al., 2017; Lei et al., 2018; Barber et al., 2019). It is therefore important to have access to efficient optimizations of full CP, for applications with strict requirements on statistical efficiency (e.g., Lei (2019)).

In our experiments, we use ICP as a time complexity baseline for our optimizations, since it is the most computationally efficient among the above techniques. We report the time complexity of the other methods in Appendix A.

Optimization of full CP classifiers. A CP is built for an ML method, by converting the method into a scoring function, the *nonconformity measure*. Informally, this function quantifies the strangeness of an example w.r.t. training data.

Makili et al. (2013) optimized CP by defining a nonconformity measure based of the Lagrangian multipliers of a trained SVM. Thanks to this, they could use an incremental version of SVM to avoid the LOO step in CP. Unfortunately, this is only a special case of SVM nonconformity measure, and being incremental is not sufficient to optimize CP in general: as we observe in this paper, in order to optimize CP, an ML method must be both incremental and decremental.

Vovk et al. (2005) optimized CP with the k-NN nonconformity measure for online learning settings when parameter k increases slowly with n; they achieved an impressive $\mathcal{O}(\log(n))$ time for 1 prediction given n training points. This method is limited to the Euclidean metric on $X = [0,1]$, or contingent on embedding the object space X in $[0,1]$. Our k-NN CP optimization works for any metric space, by exploiting a simple incremental&decremental version of k-NN we devise. Additionally, we show our idea can be used to optimize KDE CP.

Vovk et al. (2005) noticed that a linear LS-SVM nonconformity measure can be computed efficiently in the LOO step. In our work, we use the incremental&decremental LS-SVM by Lee et al. (2019) to generalize this to multiple kernels.

CP regression. The regression task in CP has been traditionally tackled separately from classification. In regression, one needs to reformulate CP (and ICP) to support an infinite label space. For ICP, this is straightforward and efficient (Papadopoulos et al., 2002). Other CP modifications for regression exist, e.g., conformal predictive distributions (Vovk et al., 2017), jackknife+, CV+ (Barber et al., 2019). As for full CP, regression is a harder goal, which was achieved only for: ridge regression (Nouretdinov et al., 2001), k-NN (Papadopoulos et al., 2011), and the Lasso (Lei, 2019). In Section 8, by using incremental&decremental learning, we produce an exact optimization of the k-NN CP regressor.

Contributions. To summarize our contributions:

- We introduce exact optimizations of full CP for the following methods: k-NN, "simplified" k-NN, KDE, and kernel LS-SVM. Each improves at least by one order of magnitude the original complexity (Table 1).
- We further use the incremental&decremental learning idea to optimize bootstrap CP by a linear factor.

- We empirically compare our techniques with i) original implementations of full CP, and ii) the most computationally efficient CP modification, ICP.
- We extend our ideas to CP regression. In particular, we improve on an optimization of the k-NN CP regressor by Papadopoulos et al. (2011), and reduce its time complexity from $\mathcal{O}(n^2m)$ to $\mathcal{O}(n\log(2n)m)$, for predicting m test objects given n training points.
- Discuss further optimization avenues for full CP.

Code to reproduce the experiments: `https://github.com/gchers/exact-cp-optimization`.

2 Preliminaries

Consider an ℓ-label classification setting, where we are given a training set of examples $Z = \{(x_1, y_1), ..., (x_n, y_n)\} \in (X \times Y)^n$ and we are asked to predict the label for a test object x.

We build a *nonconformity measure* on top of an ML method, as described in Subsection 2.1. A nonconformity measure is a real-valued function $A : (X \times Y) \times (X \times Y)^n \to \mathbb{R}$, which quantifies how much an example (x, y) "conforms to" (or is similar to) a set of training examples $\{(x_i, y_i)\}_{i=1}^n$.

For a chosen *significance level* $\varepsilon \in [0, 1]$ and nonconformity measure A, a CP classifier outputs a set $\Gamma^\varepsilon \subseteq Y$ as its prediction for test point x. CP guarantees that this prediction set contains the correct label y with at least $1-\varepsilon$ probability. Formally, if the set $\{(x_i, y_i)\}_{i=1}^n \cup \{(x, y)\}$ is exchangeable then $Pr\left(y \notin \Gamma^\varepsilon\right) \leq \varepsilon$ (Vovk et al., 2005).

Because a bound ε on the probability of error $Pr\left(y \notin \Gamma^\varepsilon\right)$ is chosen in advance, an analyst only needs to assert that Γ^ε is statistically *efficient* (i.e., it contains one or very few labels). The underlying ML method serves this purpose: the better A is, the more efficient the prediction set Γ^ε will be.

In the remainder of this section, we describe how to obtain nonconformity measures from popular ML methods, we outline the CP algorithm and its complexity, and describe ICP, the computationally-ideal baseline for our optimizations.

2.1 Nonconformity measures

We call A's output a *nonconformity score*; it takes a smaller value if example (x, y) conforms more to the training set. We give two examples of nonconformity measures.

Nearest neighbor. Let d be a metric on X. The Nearest Neighbor (NN) nonconformity measure is:

$$A((x, y); \{(x_i, y_i)\}_{i=1}^n) = \frac{\min_{i=1,...,n:y_i=y} d(x, x_i)}{\min_{i=1,...,n:y_i\neq y} d(x, x_i)}. \quad (1)$$

It is useful to think of a nonconformity measure as a scoring function determining how suitable label y is for an object x; note that this is equivalent to determining the conformity of the pair (x, y) to the training data. The NN nonconformity measure takes low values if the nearest neighbor to x that has label y is closer than its nearest neighbor with label different from y; it takes a high value otherwise. We discuss extensions of this measure in Section 3.

Nonconformity measure from generic ML methods. Let $f : X \mapsto [0, 1]^\ell$ be a classifier returning a confidence score for each of the $\ell = |Y|$ labels. We can construct a nonconformity score from f as follows:

$$A((x, y); \{(x_i, y_i)\}_{i=1}^n) = -f^y(x)\,,$$

where f is trained on $\{(x_i, y_i)\}_{i=1}^n$ and $f^y(x)$ is its score for label y. The negative sign ensures that A takes a lower value if the classifier believes y is an appropriate label for x.

2.2 Full CP classifier

Algorithm 1 CP: computing a p-value for $(x, \hat{y})$

```
1 COMPUTE_PVALUE(x, ŷ, A, Z = {(x_i, y_i)}_{i=1}^n)
2     α = A((x, ŷ); Z)
3     for i in 1, ..., n
4         α_i = A((x_i, y_i); {(x, ŷ)} ∪ Z \ {(x_i, y_i)})
5     p_(x,ŷ) = (#{i=1,...,n : α_i ≥ α} + 1) / (n + 1)
6     return p_(x,ŷ)
```

Let $Z = \{(x_i, y_i)\}_{i=1}^n$ be a training set, and x a test object. For each possible label $\hat{y} \in Y$, CP computes a p-value $p_{(x,\hat{y})}$ (Algorithm 1) based on the hypothesis that $(x, \hat{y})$ comes from the same distribution as Z; intuitively, $p_{(x,\hat{y})}$ attests on whether $\hat{y}$ is a good label for x. CP outputs the following set as its prediction: $\Gamma^\varepsilon = \{\hat{y} \in Y : p_{(x,\hat{y})} > \varepsilon\}$, for a desired value $\varepsilon \in [0, 1]$.

Time complexity of CP. Let $T_A(n)$ be the time to train A on a dataset Z of n examples, and $P_A(m)$ that of using the trained $A(\cdot; Z)$ to predict m examples. Algorithm 1 has complexity $\mathcal{O}((T_A(n) + P_A(1))n)$. If we assume the nonconformity measure should at least inspect every training point (i.e., $T_A(n) = n$), a lower bound on the complexity to compute the p-value *for one test point* is $\mathcal{O}(n^2)$.

When used for classifying a test object x in a set of labels Y, CP needs to run Algorithm 1 for every possible pairing $(x, \hat{y})$, $\hat{y} \in Y$. Therefore, the complexity becomes $\mathcal{O}((T_A(n) + P_A(1))n\ell)$, where $\ell = |Y|$. The lower bound is $\mathcal{O}(n^2\ell)$ for classifying one test point.

2.3 Inductive CP classifier

The most computationally-efficient – alas statistically inefficient, alternative to CP is inductive CP (ICP) (Vovk et al., 2005). For a parameter $t \in \{1, ..., n\}$, ICP splits the training set Z into: *proper* training set Z_{train} and *calibration*

set Z_{calib}, where $Z_{train} \cup Z_{calib} = Z$, and $|Z_{train}| = t$. Then it trains the nonconformity measure A on Z_{train}, and it computes the scores $\alpha_i = A((x_i, y_i); Z_{train})$ only for the calibration examples $(x_i, y_i) \in Z_{calib}$, instead of the entire training set; this avoids the LOO step (Lines 3-4, Algorithm 1). ICP is outlined in Appendix A.

Time complexity of ICP. Consider an ICP trained on n examples, t of which are used for the proper training set. The running time for training and calibration is $\mathcal{O}(T_A(t) + P_A(n-t))$. The time for computing the p-value for one example is $\mathcal{O}(P_A(1) + n - t)$. This becomes $\mathcal{O}((P_A(1) + n - t)\ell)$ when classifying one test object into ℓ labels.

3 Nearest neighbor nonconformity measures

We describe nonconformity measures based on the nearest neighbor principle, and introduce an optimization for their use in CP. Let d be a distance metric in the object space X.

k-NN. Equation (1) is the NN nonconformity measure, measuring the ratio of the smallest distance from examples with the same label and examples with a different label. We study a generalization of this according to the k-NN principle.

Let $\delta^j(x, S)$ be the j-th smallest distance of object x from the points in set S. The k-NN measure is (Vovk et al., 2005):

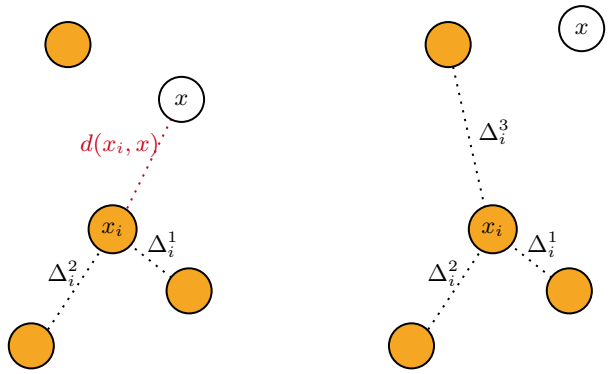
$$A((x,y);\{(x_i,y_i)\}_{i=1}^n) = \frac{\sum_{j=1}^k \delta^j(x, \{x_i \,:\, i = 1...n, y_i = y\})}{\sum_{j=1}^k \delta^j(x, \{x_i \,:\, i = 1...n, y_i \neq y\})} \,. \quad (2)$$

Simplified k-NN. Another version of the k-NN nonconformity measure, useful for anomaly detection (Laxhammar & Falkman, 2010), is defined as the nominator of Equation (2): $A((x,y);\{(x_i,y_i)\}_{i=1}^n) = \sum_{j=1}^k \delta^j(x, \{x_i \,:\, i = 1...n, y_i = y\})$. Because it only contains information for one label, we refer to it as the simplified k-NN measure.

Complexity. CP classification of m test points takes $\mathcal{O}(n^2\ell m)$ for both Simplified k-NN and k-NN. We report the derivation for all the complexities in Appendix C and Appendix D. They are summarized in Table 1.

3.1 Optimizing nearest neighbor CP

The bottleneck of Algorithm 1 is computing the nonconformity score for each training example, $\alpha_i = A((x_i, y_i); \{(x, y)\} \cup Z \setminus \{(x_i, y_i)\})$, where Z is the training set. We observe that, in order to speed this up, the nonconformity measure should be able to efficiently both learn a new example (the test example), and unlearn an example (the i-th example in the LOO step). That is, we need to devise an incremental&decremental version of k-NN.



(a) α'_i is updated (b) No update

Figure 1. Intuition behind the Simplified k-NN optimization. Training points: ●, test point: (x); $k = 3$. The nonconformity score α_i for training point x_i only depends on its k closest points. The provisional nonconformity score α'_i is updated if test point x is a k-NN of x_i (a); otherwise, no update occurs: $\alpha_i = \alpha'_i$ (b).

To this end, we get inspiration from classical techniques for LOO k-NN cross validation (e.g. Fukunaga & Hummels (1989); Hamerly & Speegle (2010)), although these are not directly applicable to our setting. The main difference is that in CP we can precompute the distances that are subsequently used to predict a test point; this enables improving the performance further.

We focus on optimizing Simplified k-NN, although the same arguments apply to k-NN. Our proposal is based on the observation that nearest neighbor measures only depend on a subset of (k) examples. We exploit this as follows. In the training phase, we precompute provisional scores:

$$\alpha'_i = A((x_i, y_i); Z \setminus \{(x_i, y_i)\}) = \sum_{j=1}^{k} \Delta_i^j \,,$$

where, for $j = 1, ..., k$:

$$\Delta_i^j = \delta^j(x_i, \{x_a \,:\, (x_a, y_a) \in Z \setminus \{(x_i, y_i)\}, y_a = y_i\}) \,.$$

Scores α'_i are provisional, because they do not account for the test example (x, y). In the prediction phase, to compute the p-value for (x, y), we update the scores as follows:

$$\alpha_i = \begin{cases} \alpha'_i - \Delta_i^k + d(x_i, x) & \text{if } \Delta_i^k > d(x_i, x) \text{ and } y_i = y \\ \alpha'_i & \text{otherwise} \,, \end{cases}$$

where Δ_i^k is the k-th smallest distance from x_i to the training examples (excluding (x_i, y_i)) with the same label as x_i. That is, we only update score α_i, associated with (x_i, y_i), if (x, y) is among its k nearest neighbors. This is illustrated in Figure 1. The cost is $\mathcal{O}(1)$.

The k-NN measure is optimized similarly, by keeping for each training example its k best distances from both objects with the same label and from those with a different label.

Complexity. For both measures, the training cost is $\mathcal{O}(n^2)$. Classifying m test examples is $\mathcal{O}(n\ell m)$.

4 Kernel Density Estimation

For a kernel function K, the Kernel Density Estimation (KDE) nonconformity measure is:

$$A((x,y);\{(x_i,y_i)\}_{i=1}^n) = -\frac{1}{n_y h^p}\sum_{x_i:y_i=y} K\left(\frac{x-x_i}{h}\right),$$

where $n_y = \#\{i=1,...,n \,:\, y_i = y\}$, h is the bandwidth, and p is the objects' dimensionality.

Complexity. If computing the kernel for one object is P_K, CP classification takes $\mathcal{O}(P_K n^2 \ell m)$.

4.1 Optimizing KDE CP

We use a similar idea to that of our k-NN optimization; however, in this case A depends on all the training points, not just a subset. To the best of our knowledge, this incremental&decremental adaptation of KDE is also novel. For training, we compute preliminary scores:

$$\alpha'_i = \sum_{x_j:y_j=y_i} K\left(\frac{x_i-x_j}{h}\right) \quad i=1,...,n\,.$$

To calculate the p-value for an example (x,y) in the test phase, we update the scores as follows:

$$\alpha_i = \begin{cases} -\frac{1}{n_y h^d}\left(\alpha'_i + K\left(\frac{x-x_i}{h}\right)\right) & \text{if } y_i = y \\ -\frac{1}{n_y h^d}\alpha'_i & \text{otherwise}\,. \end{cases}$$

Complexity. Training takes $\mathcal{O}(P_K n^2)$. CP classification runs in $\mathcal{O}(P_K n\ell m)$.

5 Least Squares Support Vector Machine

Assume $Y = \{-1, 1\}$. Consider a feature map $\phi : X \to F$. The least-squares SVM (LS-SVM) regressor, defined by ϕ and a vector w, returns a prediction for an object x as: $w^\top \phi(x)$. The model w is trained with Tikhonov regularization (ridge regression); details in Appendix B. We define the nonconformity measure for the LS-SVM regressor as:

$$A((x,y);\{(x_1,y_1),...,(x_n,y_n)\}) = -yf(x)\,;$$

it takes high values if the prediction $f(x)$ is different (in sign) from y. Extension of this to $\ell = |Y| > 2$ can be done via one-vs-rest approaches (e.g., Vovk et al. (2005)).

Complexity. Depending on algorithm choices, training LS-SVM takes n^ω, $\omega \in [2,3]$. CP LS-SVM takes $\mathcal{O}(n^{\omega+1}\ell m)$.

5.1 Optimizing LS-SVM CP

We exploit recent work by Lee et al. (2019), which enables exact incremental and decremental learning of LS-SVM. Given a trained model w, their proposal enables updating w by adding/removing an example in time $\mathcal{O}(q^3)$, where q is the dimensionality of the feature space F (Appendix B).

We apply this for optimizing LS-SVM CP. In the training phase, we learn the model w on the training data. Then, to compute the nonconformity score for an example (x_i, y_i), we: i) update the model with the test example by using the approach by Lee et al., ii) make a prediction for (x_i, y_i).

Complexity. Training LS-SVM takes $\mathcal{O}(n^\omega)$, for $\omega \in [2,3]$ (one-off cost). CP classification is $\mathcal{O}(q^3 n\ell m)$.

Discussion. Other options are possible for optimizing SVM nonconformity measures. Cauwenberghs & Poggio (2001) proposed an incremental&decremental version of SVM, which differently from the one we used has a larger memory footprint. Another promising avenue for optimization is the classical linear SVM formulation using coordinate-descent, in combination with incremental updates (Tsai et al., 2014).

6 Bootstrapping methods

Let integer $B > 1$ be a hyperparameter, and select a *base classifier* (e.g., decision tree). In bootstrapping, the training data $Z = \{(x_1,y_1),...,(x_n,y_n)\}$ is sampled B times with replacement to produce B bootstrap samples, $Z_1,...,Z_B$. On each sample we fit the base classifier, obtaining an ensemble of B classifiers $(g_1,...,g_B)$, which we jointly denote with $f : X \to [0,1]^\ell$, $\ell = |Y|$.

Classifier f outputs a confidence vector, $f(x) \in [0,1]^\ell$, over the labels. The y-th element of this vector, denoted by $f^y(x)$, is computed as the normalized count of classifiers g_i that predict y. That is:

$$f^y(x) = \frac{1}{B}\#\{i=1,...,B \,:\, g_i(x) = y\} \quad y \in Y\,.$$

We define the bootstrapping nonconformity measure as:

$$A((x,y);\{(x_1,y_1),...,(x_n,y_n)\}) = -f^y(x)\,.$$

Complexity. Let $T_g(n)$ be the time needed to train the base classifier on n training points, and $P_g(m)$ its cost to predict m points. Bootstrap CP runs in $\mathcal{O}((T_g(n)+P_g(1))Bn\ell m)$.

6.1 Optimizing bootstrap CP

Standard bootstrap CP requires training a bootstrap ensemble for each training example (x_i, y_i) and one for the test example (x, y); this entails creating, for each example, B bootstrap samples that do not contain that example. The

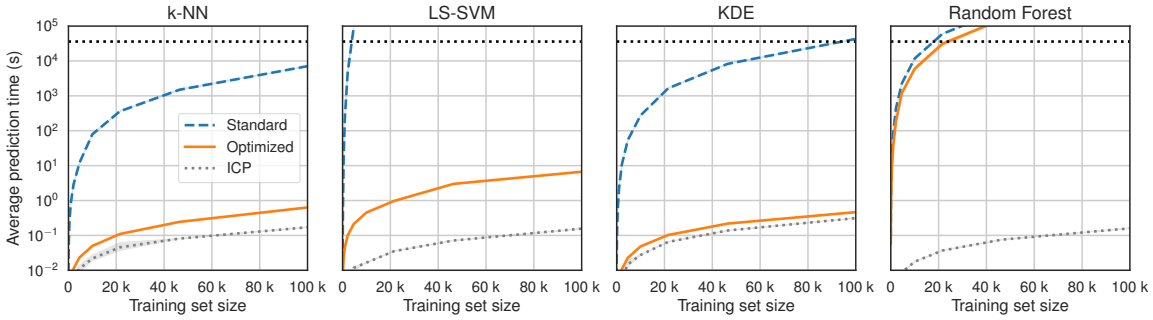


Figure 2. Comparison between the standard and optimized full CP. ICP serves as a baseline for these measurements. Prediction time for one test point w.r.t. the size of training data. Black dashed line is the experiment timeout (10 hours).

optimization we propose maintains the spirit of bootstrap, although it may lead to different results from the standard version because of changes in the sampling strategy.

We first explain the basic idea for training and prediction, and then improve it with two remarks. Let "$*$" be a placeholder for the test point (x, y), which is unavailable during training, and let $Z^* = Z \cup \{*\}$ be the augmented training set. For a number $B' > B$ to be later specified, we create B' bootstrap samples of Z^*, denoted $\{Z_1^*, ..., Z_{B'}^*\}$. We continue creating samples until, for every point $(x_i, y_i) \in Z^*$, there are at least B bootstrap samples that *do not* contain (x_i, y_i); that is, we increase the number of samples B' until $\#\{b = 1, ..., B' : (x_i, y_i) \notin Z_b^*\} \geq B$ for all $(x_i, y_i) \in Z^*$. This ensures that each training point (and the placeholder test point) have at least B bootstrap samples.[1]

Let $E_i = \{b = 1, ..., B' : (x_i, y_i) \notin Z_b^*\}$ be the samples associated with (x_i, y_i), and $E = \{b = 1, ..., B' : * \notin Z_b^*\}$ the ones associated with the (placeholder) test example. In the prediction phase, we compute a prediction for test point (x, y) by using the base classifiers trained on the bootstrap samples in E. We make the prediction for a training point (x_i, y_i) in the LOO procedure of CP as follows: i) in E_i's bootstrap samples, replace the placeholder $*$ with the test point (x, y), ii) train the base classifiers on the samples from E_i and compute a prediction for (x_i, y_i).

Remarks. The procedure explained so far preemptively samples B bootstraps for each point. We apply the following improvements. Because some bootstrap samples $Z_b^* \in E_i$ associated with (x_i, y_i) do not contain the placeholder $*$, in the training phase we: i) pretrain the base classifiers $g_b(x)$ on them, and ii) compute their predictions for (x_i, y_i). This saves up considerable time in the prediction phase. The optimized bootstrap algorithm is listed in Appendix B.

Complexity. Optimized CP classification for m test points is $\mathcal{O}((T_g(n) + P_g(1))(1 - e^{-1})B\ell m)$, a factor $(1 - e^{-1}) \approx 0.632$ speed up on the standard one. The speed up of this optimization is not as prominent as our other proposals. However, we suspect one can further improve bootstrap CP for base classifiers that support incremental&decremental learning (Section 9). We leave this to future work.

[1] If at the end of the procedure an example has more than B, we can truncate them to B to save up on computational resources.

7 Empirical evaluation

We compare the running time of the original and optimized CP, using ICP as a baseline. We detail hardware, precautions taken to ensure the fidelity of the measurements, and hyperparameters in Appendix E. We instantiate bootstrap CP to Random Forest.

7.1 Comparison between standard and optimized CP

Setup. In our experiments, the data distribution is irrelevant. We generate data for a binary classification problem with 30 features, by using the `make_classification()` routine of the `scikit-learn` library. (In Appendix G, we further compare CP and ICP on the `MNIST` dataset.)

For every training size n, chosen in the space $[10, 10^5]$, we train the CP with a nonconformity measure, and use it to predict 100 test points. We set a timeout of 10 hours, which is verified after the prediction of every test point; therefore, the timeout may be exceeded if the prediction for a point has already started. We measure both the training time and the average prediction time for a test point. Each experiment is repeated for 5 different initialization seeds.

Prediction time. Figure 2 shows the comparison between standard and optimized CP. Results confirm the complexity we derived analytically. For 100k training points, the optimized k-NN CP ensures a prediction in 0.63 seconds, whilst the respective unoptimized version takes roughly 2 hours for the same prediction. Since k-NN and Simplified k-NN behave very similarly, results for the latter are in Appendix F. The largest speed up is with LS-SVM: the

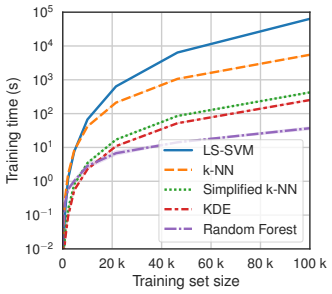


Figure 3. Training time of optimized CP.

optimized version has a running time of 0.21 seconds; the standard implementation takes on average more than 24.5 hours for 1 prediction. Our bootstrap CP optimization only gives a marginal improvement over the original implementation. For $n = 46415$, optimized Random Forest takes 43 hours for one prediction, the standard one 82 hours.

Comparison with ICP. We use ICP as a baseline. For a parameter $t \in \{1, ..., n\}$, ICP trains the nonconformity measure on a subset of t examples, and computes the scores for the remaining $n - t$. We fix $t/n = 0.5$.

As expected, results (Figure 2) show that ICP is strictly faster than the optimized CP methods: e.g., when trained on 100k examples, LS-SVM takes 6.68 seconds per prediction, while LS-SVM ICP takes 0.16 seconds; the worst performing is Random Forest, which as seen above improves CP only by a linear factor. Nevertheless, in some cases ICP and optimized CP have the same magnitude: for KDE, ICP takes 0.31 seconds, optimized CP 0.46 seconds. In other words, our CP optimization seems to perform comparably well to ICP on reasonably large datasets.

This reveals a better trade-off between computational-statistical efficiency in conformal inference: if one's priority is speed, they can use ICP, or other CP alternatives; however, if they can sacrifice computational time, they can get full CP predictions and yet scale to real-world data.

7.2 Training time

CP with the optimized nonconformity measures incurs into a training time, while standard CP does not. We compare the training time of the optimized measures in Figure 3.

We observe that LS-SVM has the highest training time, whilst Random Forest the lowest. We also notice that the training time is a reasonable price to pay in practice. In a batch classification setting with 100k training and 20k test examples, optimized k-NN CP would take 2.2 hours for training and 3.3 hours for prediction. Standard k-NN CP would have no training time, but its prediction routine would run for 9.3 years to obtain the same solution.

It may be possible to speed up our techniques even further via approximate incremental&decremental learning techniques. We leave this to future work (Section 9).

8 Large Y and extension to regression

The classification algorithms for CP (Algorithm 1) and ICP (Algorithm 2) are clearly unfeasible for a very large Y: they both require repeating the calculations for each $y \in Y$.

Things are different for regression, where we assume a total order on Y. In this case, one can avoid the $\ell = |Y|$ term in the cost of both CP and ICP (Vovk et al., 2005). Indeed, it is possible to find the intervals of Y where the p-value $p_{(x,\tilde{y})}$ exceeds ε, without having to try all values $\tilde{y} \in Y$. In ICP, this can be done efficiently for general regressors.

As for full CP, this optimization is harder, as one needs to update the intervals of Y for each training point when a new point arrives. Full CP regression was optimized in this sense for k-NN (Papadopoulos et al., 2011), ridge regression (Nouretdinov et al., 2001), and Lasso (Lei, 2019). Ndiaye & Takeuchi (2019) recently proposed a general method leading to approximate but statistically valid CP regressors.

Since the above full CP regression methods do not exploit incremental&decremental ideas, we suspect they can be optimized further. We show this is possible for k-NN.

8.1 Improving the k-NN CP regressor

The full k-NN CP regressor works as follows. Fix an hyperparameter $k > 0$. Let $\tilde{y} \in Y$ be a candidate label (not to be defined explicitly) for test object x. Define the nonconformity score for the i-th training example (x_i, y_i) as:

$$\alpha_i = \alpha_i(\tilde{y}) = |a_i + b_i\tilde{y}| \ ,$$

where, for $i = 1, ..., n$:

$$a_i = \begin{cases} y_i - \frac{1}{k}\sum_{j=1}^{k-1} y_{(j)}(x_i) & \text{if } x \text{ is one of } x_i\text{'s } k \text{ NNs} \\ y_i - \frac{1}{k}\sum_{j=1}^{k} y_{(j)}(x_i) & \text{otherwise} \, , \end{cases}$$

$$b_i = \begin{cases} -\frac{1}{k} & \text{if } x \text{ is one of } x_i\text{'s } k \text{ NNs} \\ 0 & \text{otherwise} \, ; \end{cases}$$

here $y_{(j)}(x_i)$ is the label of the j-th nearest neighbor of x_i in the training set $Z \setminus (x_i, y_i)$. For the test example x, we set $a = -1/k \sum_{j=1}^{k} y_{(j)}(x)$, $b = 1$. The p-value is:

$$p_{(x,\tilde{y})} = \frac{\#\{i = 1, ..., n \ : |a_i + b_i\tilde{y}| \geq |a + b\tilde{y}|\}}{n + 1} \, .$$

The optimization idea by Papadopoulos et al. (2011) is based on the fact that, in order to find an interval of Y for which $p_{(x,\tilde{y})} > \varepsilon$, it suffices to find the points $\tilde{y} \in Y$ for which $c(\tilde{y}) = \alpha_i(\tilde{y}) - \alpha(\tilde{y})$ changes. This can be done efficiently,

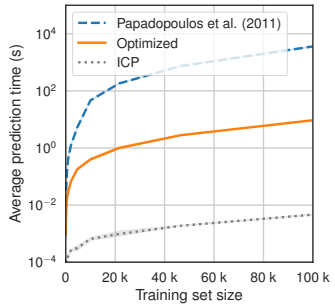


Figure 4. Time comparison of k-NN CP regression: method by Papadopoulos et al. (2011), our optimization, and ICP (baseline).

by only looking at most at $2n$ points. The time complexity of one prediction is $\mathcal{O}(n^2 + 2n\log(2n))$, where the term $\mathcal{O}(n^2)$ comes from computing the k nearest neighbors of each training point, and $\mathcal{O}(2n\log(2n))$ comes from sorting the critical points of $c(\tilde{y})$ (required by the above algorithm).

Optimization via incremental&decremental learning. The method by Papadopoulos et al. (2011) can be further improved via the incremental&decremental k-NN algorithm we proposed in this paper. We reduce the $\mathcal{O}(n^2)$ term as follows. In the training phase, we: i) precompute the pairwise distances of the training points in Z, ii) and precompute temporary values a'_i and b'_i, for $i = 1, ..., n$. Specifically, we let $a_i = y_i - \frac{1}{k}\sum_{j=1}^{k} y_{(j)}(x_i)$ and $b_i = 0$, as if the (yet unknown) test example x did not contribute to their values. When making a prediction for x, we: iii) compute its distance from the elements of Z (takes $\mathcal{O}(n)$), and iv) update those a'_i and b'_i such that x is one of the k- nearest neighbors of (x_i, y_i). Then we proceed as before.

Even in this setting, using an incremental&decremental version of the nonconformity measure enables us to reduce the prediction complexity by almost one order of magnitude. Predicting m test examples reduces from $\mathcal{O}((n^2 + 2n\log(2n))m)$ to $\mathcal{O}(2n\log(2n)m)$.

Empirical evaluation. We compare full k-NN CP regression (Papadopoulos et al., 2011) with our optimization via incremental&decremental learning. As a baseline we use ICP k-NN regression, whose complexity is $\mathcal{O}(tm))$, where $t \in \{2, ..., n-1\}$ is the size of the proper training set, and m the number of test points. We generate regression examples from $X \times Y = \mathbb{R}^{30} \times \mathbb{R}$ with `scikit-learn`'s `make_regression()` function. We vary $n \in [10, 10^5]$, and measure the average prediction time across 100 test points. Each experiment is repeated for 5 random seeds, and confidence intervals are plotted.

Figure 4 shows that our optimization largely outperforms the previous version of full k-NN CP regression by Papadopoulos et al. (2011); the cost for one prediction with 100k training points decreases from 1 hour to 9.3 seconds. ICP outperforms both, taking roughly 4.6 ms. We remark, however, that ICP was observed to have a strictly weaker statistical power in regression (Papadopoulos et al., 2011).

Discussion. We expect that our LS-SVM CP optimization (Section 5) can be readily applied to speed up the full CP regressor based on ridge regression (Nouretdinov et al., 2001). We leave this, and the optimization of other CP regressors using incremental&decremental ideas, to future work.

9 Discussion and conclusion

Full CP is computationally expensive because of its main routine (Algorithm 1), which runs a leave-one-out (LOO) procedure on the ML method (*nonconformity measure*) that it wraps. In this paper, we show that if a nonconformity measure can be designed to learn and unlearn one example efficiently (i.e., it can be trained incrementally&decrementally), this can speed up considerably CP classification. Concretely, we improved k-NN, KDE, and kernel LS-SVM CP classifiers by at least one order magnitude, and bootstrap CP by a linear factor. Furthermore, we exploited these ideas to further optimize k-NN CP regression. Our work makes it feasible to run full CP on large datasets.

We discuss how our optimizations are readily applicable to other tasks (e.g., clustering, change-point detection), and future directions for CP optimization.

Extensions to more learning tasks. In addition to classification and regression, CP is used for tasks such as anomaly detection (Laxhammar & Falkman, 2010), clustering, and sequence prediction (Cherubin & Nouretdinov, 2016). Because all these techniques are based on computing a p-value via Algorithm 1, our optimizations are immediately applicable. For example, conformal clustering (Cherubin et al., 2015) with k-NN CP costs $\mathcal{O}(n^2q^p)$, where q is the length of a square grid constructed around p-dimensional training points. With our optimization, the cost becomes $\mathcal{O}(nq^p)$. (Usually, $p = 2$, by using dimensionality reduction.)

CP has applications to online learning (e.g., change-point detection (Vovk et al., 2003)). At step $n + 1$, the algorithm trains on examples $\{(x_i, y_i)\}_{i=1}^{n}$, makes a prediction for x_{n+1}, and learns the true label y_{n+1}. Adapting our optimizations to this setting is trivial: it suffice to incrementally learn the new example (x_{n+1}, y_{n+1}) after prediction, which is efficient for k-NN, KDE and LS-SVM. This has a considerable speed-up. For example, an IID test by Vovk et al. (2003), which has further applications to feature selection (Cherubin et al., 2018), requires to incrementally compute a p-value for the $(n + 1)$-th point given $\{(x_i, y_i)\}_{i=1}^{n}$. With k-NN CP, this costs $\mathcal{O}(n^3)$; our method reduces it to $\mathcal{O}(n^2)$ (Appendix C). Unfortunately, this is not efficient for bootstrap; we leave its further optimization to future work.

The umbrella of conformal inference also includes methods such as Venn Predictors (VP), which give analogous guarantees to CP, but for the calibration of probabilistic predictions. Future work may investigate whether VP can be optimized with similar techniques to the ones we proposed.

Boosting and gradient descent. We hope our work will inspire optimization techniques for more nonconformity measures. We foresee as particularly challenging the optimization of methods such as boosting and gradient descent. For both techniques, the contribution of a training example depends on previous examples. Hence, unlearning an example has a high cost, as it requires updating the contributions of all the examples that came after. We suggest recent work on unlearning methods may help to achieve this goal.

Approximations. Another natural avenue is to use approximate incremental&decremental learning techniques. For example, by bounding the contribution of each point it may be possible to achieve very computationally efficient methods with little cost on statistical efficiency.

Exploiting multiple CPUs, GPUs. A further direction is to study how to exploit a GPU or multiple CPUs to speed up CP. Towards this goal, we conducted a preliminary comparison between parallel and sequential implementations of CP and optimized CP (Appendix H); CP and optimized CP are parallelized in the same way. Results show that, for a small dataset (5k examples) standard CP benefits from parallelization, while optimized CP does not substantially. Surprisingly, in this case optimized k-NN is even faster without parallelization, although it does benefit for larger datasets. More research is needed to determine the best parallelization strategies for CP, both from an algorithmic and implementational level. We leave this, and the study of GPUs for CP, to future work.

Acknowledgements

We are grateful to Vladimir Vovk for interesting discussion.

References

Barber, R. F., Candes, E. J., Ramdas, A., and Tibshirani, R. J. Predictive inference with the jackknife+. *arXiv preprint arXiv:1905.02928*, 2019.

Carlsson, L., Eklund, M., and Norinder, U. Aggregated conformal prediction. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*, pp. 231–240. Springer, 2014.

Carlsson, L., Bendtsen, C., and Ahlberg, E. Comparing performance of different inductive and transductive conformal predictors relevant to drug discovery. In *Conformal and Probabilistic Prediction and Applications*, pp. 201–212, 2017.

Cauwenberghs, G. and Poggio, T. Incremental and decremental support vector machine learning. In *Advances in neural information processing systems*, pp. 409–415, 2001.

Cherubin, G. and Nouretdinov, I. Hidden markov models with confidence. In *Symposium on Conformal and Probabilistic Prediction with Applications*, pp. 128–144. Springer, 2016.

Cherubin, G., Nouretdinov, I., Gammerman, A., Jordaney, R., Wang, Z., Papini, D., and Cavallaro, L. Conformal clustering and its application to botnet traffic. In *International Symposium on Statistical Learning and Data Sciences*, pp. 313–322. Springer, 2015.

Cherubin, G., Baldwin, A., and Griffin, J. Exchangeability martingales for selecting features in anomaly detection. In *Conformal and Probabilistic Prediction and Applications*, pp. 157–170. PMLR, 2018.

Fisch, A., Schuster, T., Jaakkola, T., and Barzilay, R. Efficient conformal prediction via cascaded inference with expanded admission. In *International Conference on Learning Representations (ICLR)*, 2021.

Fukunaga, K. and Hummels, D. M. Leave-one-out procedures for nonparametric error estimates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11 (4):421–423, 1989.

Gupta, C., Kuchibhotla, A. K., and Ramdas, A. K. Nested conformal prediction and quantile out-of-bag ensemble methods. *arXiv preprint arXiv:1910.10562*, 2019.

Hamerly, G. and Speegle, G. Efficient model selection for large-scale nearest-neighbor data mining. In *British National Conference on Databases*, pp. 37–54. Springer, 2010.

Laxhammar, R. and Falkman, G. Conformal prediction for distribution-independent anomaly detection in streaming vessel data. In *Proceedings of the first international workshop on novel data stream pattern mining techniques*, pp. 47–55, 2010.

LeCun, Y., Cortes, C., and Burges, C. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

Lee, W.-H., Ko, B. J., Wang, S., Liu, C., and Leung, K. K. Exact incremental and decremental learning for LS-SVM. In *2019 IEEE International Conference on Image Processing (ICIP)*, pp. 2334–2338. IEEE, 2019.

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

Lei, J., G'Sell, M., Rinaldo, A., Tibshirani, R. J., and Wasserman, L. Distribution-free predictive inference for regression. *Journal of the American Statistical Association*, 113(523):1094–1111, 2018.

Linusson, H., Johansson, U., Boström, H., and Löfström, T. Efficiency comparison of unstable transductive and inductive conformal classifiers. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*, pp. 261–270. Springer, 2014.

Makili, L., Vega, J., and Dormido-Canto, S. Incremental support vector machines for fast reliable image recognition. *Fusion Engineering and Design*, 88(6-8):1170–1173, 2013.

Ndiaye, E. and Takeuchi, I. Computing full conformal prediction set with approximate homotopy. *arXiv preprint arXiv:1909.09365*, 2019.

Nouretdinov, I., Melluish, T., and Vovk, V. Ridge regression confidence machine. In *ICML*, pp. 385–392. Citeseer, 2001.

Papadopoulos, H., Proedrou, K., Vovk, V., and Gammerman, A. Inductive confidence machines for regression. In *European Conference on Machine Learning*, pp. 345–356. Springer, 2002.

Papadopoulos, H., Vovk, V., and Gammerman, A. Regression conformal prediction with nearest neighbours. *Journal of Artificial Intelligence Research*, 40:815–840, 2011.

Tsai, C.-H., Lin, C.-Y., and Lin, C.-J. Incremental and decremental training for linear classification. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 343–352, 2014.

Vovk, V. Cross-conformal predictors. *Annals of Mathematics and Artificial Intelligence*, 74(1-2):9–28, 2015.

Vovk, V., Nouretdinov, I., and Gammerman, A. Testing exchangeability on-line. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pp. 768–775, 2003.

Vovk, V., Gammerman, A., and Shafer, G. *Algorithmic learning in a random world*. Springer Science & Business Media, 2005.

Vovk, V., Fedorova, V., Nouretdinov, I., and Gammerman, A. Criteria of efficiency for conformal prediction. In *Symposium on Conformal and Probabilistic Prediction with Applications*, pp. 23–39. Springer, 2016.

Vovk, V., Shen, J., Manokhin, V., and Xie, M.-g. Nonparametric predictive distributions based on conformal prediction. In *Conformal and Probabilistic Prediction and Applications*, pp. 82–102. PMLR, 2017.