Coreset-based Conformal Prediction for Large-scale Learning

Nery Riquelme-Granada Nery.RiquelmeGranada.2016@live.rhul.ac.uk
Khuong An Nguyen Khuong.Nguyen@rhul.ac.uk
Zhiyuan Luo Zhiyuan.Luo@rhul.ac.uk
Computer Science Department, Royal Holloway, University of London
Egham, Surrey TW20 0EX
United Kingdom

Editor: Alex Gammerman, Vladimir Vovk, Zhiyuan Luo and Evgueni Smirnov

Abstract

As the volume of data increase rapidly, most traditional machine learning algorithms become computationally prohibitive. Furthermore, the available data can be so big that a single machine's memory can easily be overflown. We propose Coreset-Based Conformal Prediction, a strategy for dealing with big data by applying conformal predictors to a weighted summary of data - namely the coreset. We compare our approach against standalone inductive conformal predictors over three large competition-grade datasets to demonstrate that our coreset-based strategy may not only significantly improve the learning speed, but also retains predictions validity and the predictors' efficiency.

Keywords: Coreset, logistic regression, importance sampling, conformal predictors.

1. Introduction

When facing a learning problem, it is highly desirable that the learner receives data that accurately represent some underlying process we want to learn about. Furthermore, such an ideal scenario is more likely to occur as there is more data available to the learner. However, the blessing of large-scale data availability comes with severe computational constraints: firstly, the computing power available might be easily overshadowed by the volume of information at hand; secondly, the data may quickly overflow the local storage.

We are interested in studying this trade-off in the context of *Conformal Prediction*, where our learner outputs *valid* fine-grained prediction sets with tight bounds on the number of prediction errors (Gammerman and Vovk (2007)). Conformal Predictors (CPs) are known to be computationally intensive, even in their inductive version. Thus, scaling these algorithms to the big data setting is an active research area.

In this work we give an alternative solution for analysing massive datasets using Conformal Predictors: we propose the use of *coresets*, a data-summarising framework, to produce a small *weighted* set of data that provably correctly approximates the original big data (Phillips (2016)). After we obtain a coreset, we can safely discard the original dataset and proceed to compute Conformal Predictors over the weighted data summary, dramatically reducing storage and computational overheads. We call this strategy *Coreset-Based Conformal Prediction* (CBCP) and we shall investigate its application to the problem of classifying labeled objects using Logistic Regression.

The contributions of our work can be summarised as follows:

- **Coreset-based Conformal Prediction.** We bring the idea of coresets to Conformal Prediction. Our strategy provides, as we shall see in Section 4, an interesting boost to Conformal Predictors' computational efficiency.
- **Empirical analysis.** We provide detailed empirical evaluation of our strategy. Particularly, we evaluate the extent to which using coresets affects validity and efficiency of Conformal Predictors. We show in our study that when Conformal Predictors are computed over coresets, their validity and efficiency are retained *i.e.* CPs over the coreset and CPs over the full data give virtually the same number of prediction errors and the same efficiency scores.

The paper is organised according to the following structure. In Section 2 we briefly discuss the Logistic Regression learning problem and Conformal Prediction. Section 3 is devoted to coresets: their definition, construction and properties. We present our experimental results and discussions in Section 4. Section 5 overviews some related work. Finally, we summarise our work and outline future research directions in Section 6.

2. Binary logistic regression and Conformal Prediction

This section overviews the logistic regression problem as well as the concepts involved in Conformal Prediction (i.e. transductive Conformal Prediction and inductive Conformal Prediction). Notice that a logistic regression model will be used as the underlying predictor for Conformal Prediction.

2.1. The learning problem

We are interested in the problem of binary classification using Logistic Regression (LR), which is a well-known instance of a generalised linear model. We choose to work with the binary version because the problem exposition is simplified and generally it is straightforward to extend the binary results to multi-class results by procedures like "one-against-all" or "one-against-one" (Milgram et al. (2006)).

Formally, we are given a dataset $\mathcal{D} := \{(x_n, y_n)\}_{n=1}^{N}$, where $x_n \in \mathbb{R}^d$ is a feature vector and $y_n \in \{-1, 1\}$ is its corresponding label. For LR, the likelihood of observation y_n, given some parameter $\theta \in \mathbb{R}^{d+1}$, is $p_{logistic}(y_n = 1|x_n; \theta) := 1/(1 + \exp(-x_n \cdot \theta))$ and

$$p_{logistic}(y_n = -1|x_n; \theta) := 1 - \frac{1}{(1 + \exp(-x_n \cdot \theta))} = \frac{\exp(-x_n \cdot \theta)}{(1 + \exp(-x_n \cdot \theta))} = \frac{1}{(1 + \exp(x_n \cdot \theta))}.$$

Therefore we can set $p_{logistic}(y_n|x_n; \theta) := 1/(1 + \exp(-y_n x_n \cdot \theta))$ for any y_n and finally define the log-likelihood function $LL_N(\theta|\mathcal{D})$ as specified in Shalev-Shwartz and Ben-David (2014):

$$LL_N(\theta|\mathcal{D}) := \sum_{n=1}^{N} \ln p_{logistic}(y_n|x_n; \theta) = -\sum_{n=1}^{N} \ln(1 + \exp(-y_n x_n \cdot \theta)) \tag{1}$$

which is the objective function for the LR problem. The optimal parameter $\hat{\theta}$ can be estimated by maximising $LL_N(\theta|\mathcal{D})$. In other words, we want to minimise $\mathcal{L}_N(\theta|\mathcal{D}) := \sum_{n=1}^{N} \ln(1+\exp(-y_n x_n \cdot \theta))$ over all $\theta \in \mathbb{R}^{d+1}$. Namely, the optimisation problem is defined as:

$$\arg\min_{\theta \in \mathbb{R}^{d+1}} \mathcal{L}_N(\theta|\mathcal{D}). \tag{2}$$

Our work relies on the assumption that we have *redundancy* in data and hence not all points in the dataset are equally important from the learning perspective. That is, there are points in $\mathcal{D}$ that contribute more to Equation 2. This observation constitutes the base for our approach: we will identify those important points for the LR problem, assign weights to them and discard the rest of the points. To such subset of the data we call a coreset, and we will carefully explain its properties in Section 3.

2.2. Conformal Predictors

We now complete our problem definition by introducing the learning framework we are interested in studying. We first define Conformal Prediction in the transductive setting; then we expand the definitions to the inductive scenario.

Conformal Predictors (Vovk et al. (2005)) are learning algorithms that evaluate conformity. In essence, CPs compute, for a given object we want to classify, prediction *sets* containing a *p-value* for each possible label the object can have. The p-values provide a powerful reliability measure that quantifies the likelihood that we find an object with a given label under the assumption that all objects are randomly and independently drawn from the same probability distribution (IID assumption).

We use in this subsection a more compact notation for representing feature vectors and labels: we define $z_i = (x_i, y_i)$ to be the *i-th* example of some sequence Z^N composed of N examples.

Let N be any natural number, a non-conformity measure $\mathcal{A}$ assigns to every sequence $(z_1, z_2, ..., z_N)$ of N examples a sequence $(\alpha_1, \alpha_2, ..., \alpha_N)$ of N real-valued nonconformity scores which is equivariant with respect to permutations (Balasubramanian et al. (2014)). We define the Conformal Predictor, determined by the nonconformity measure $\mathcal{A}$, as in Vovk et al. (2016):

$$\Gamma^{\epsilon}(z_1, z_2 ... z_N, x) := \{y : p^y > \epsilon\}, \tag{3}$$

where for each label $y \in Y$ *i.e.* $Y = \{-1, 1\}$ in our case, the *p-value* p^y is defined by

$$p^y = p^y(z_1, z_2 ... z_N, x) := \frac{|\{i = 1, 2 ... N+1 : \alpha_i^y \geq \alpha_{N+1}^y\}|}{N+1} \tag{4}$$

Finally, we define the sequence of non-conformity scores:

$$(\alpha_1^y, \alpha_2^y, ..., \alpha_{N+1}^y) := \mathcal{A}(z_1, z_2, ..., z_N, z) \tag{5}$$

where $z = (x, y)$.

The CP framework uses existing machine learning algorithms, which are commonly referred as *simple predictors*, as subroutines. In fact, simple predictors constitute the underlying algorithm for defining $\mathcal{A}$. See Vovk et al. (2005) for a rich list of algorithms and strategies for computing nonconformity measures.

The way we compute the prediction set in Equation (3) might change depending on the nature of the problem at hand. Let x_{N+1} be the feature vector of an example that we just obtained from a sequence of examples. For our classification problem, we compute Equation (4) for each pair $(x_{N+1}, y)\ \ y \in Y$ and filter the $|Y|$ p-values using Equation (3). We then repeat the same procedure for each new example.

2.2.1. Inductive Conformal Predictors

By inspecting the above definitions of Conformal Predictors, it is not hard to see that these algorithms are designed for the online setting. In fact, they enjoy stronger validity when run in a purely online fashion. Intuitively, as the training sequence gets larger, they make better predictions since they have more "experience" available. This advantage does not come for free, though: as mentioned previously, the whole framework has to be recomputed from scratch for each new example. Clearly, these *transductive* algorithms soon become computationally prohibitive, which makes them ill-suited for relatively large data.

Inductive Conformal Predictors (ICPs) (Vovk (2012)) are *inductive* variants of Conformal Predictors designed to extend the applicability of their transductive counterparts by first, easing the computational overhead; and second, relaxing the pure-online constraint. The price we pay for ICPs is weakened validity and decreased efficiency *i.e.* larger and hence less informative prediction sets. For a detailed explanation on the distinction between the notions of *transductive* and *inductive* learning, we refer the reader to Vapnik's work (Vapnik (1998)).

To compute inductive Conformal Predictors, we need to split the training set $(z_1, z_2, ..., z_N)$ into two parts:

- $(z_1, z_2, ..., z_m)$, the proper training set of size $m < N$;
- $(z_{m+1}, z_{m+2}, ..., z_N)$, the calibration set of size $N - m$.

Generally, we would expect to have a big training set and a considerably smaller calibration set. We now need to adapt the definitions given in Section 2.2 to the inductive setting: we define the nonconformity measure as $\mathcal{A} : Z^m \times Z \to \mathbb{R}$ and re-write Definition (4) as:

$$p^y := \frac{|\{i = m+1, m+2, ..., N+1 : \alpha_i^y \geq \alpha_{N+1}^y\}|}{N - m + 1} \tag{6}$$

with the corresponding non-conformity scores:

$$\alpha_i = \mathcal{A}(z_1, z_2, ..., z_m, z_i), i = m+1, m+2, ..., N;$$
$$\alpha_{N+1}^y = \mathcal{A}(z_1, z_2, ..., z_m, z_{N+1}).$$

The intuition behind inductive Conformal Predictors is that they measure how well a new example conforms to the training set with respect to the calibration set.

From the above definitions, we can see why ICPs save a great deal of computing time: they only compute the NCM for each example in the calibration set once. Then, for each new example, they only need to compute one NCM score in order to compare it to the current pool of scores. Hence, ICPs are better-suited for large-scale learning problems compared to CPs. We shall see, however, that our coreset-based strategy can drastically improve the computing time achieved by ICPs.

3. Coreset-Based Conformal Predictors

Having discussed the background concepts of transductive and inductive CP, we are now in a good position to present our coreset-based approach to improve the computing time of the latter. We present first the intuition and motivation behind summarising data. Then we present the specific techniques we use in our strategy: coresets.

3.1. Summarising data

Data summarising has been of great interest in machine learning in the last decades. As we previously mentioned, having more data means access to more information from which, hopefully, we will get a better understanding of an underlying truth. We know that learning becomes harder as data grows; this is because many machine learning algorithms use computationally expensive numerical solvers to optimise an underlying objective function. Given that not all data points in a dataset are equally relevant to this optimisation problem, substantial research effort has been put in identifying the portion of data that is more important from the optimisation point of view. The set constructed with the more important portion of data can be interpreted as a summary of the original dataset since it contains sufficient information to provide a (provably) *good* solution for the optimisation problem. It is common to refer to a summary as a "compression" of the original data (Feldman et al. (2013)), to emphasise the fact that the portion of important points is small compared to the original full data.

Under this paradigm, the computation is divided in two intuitive steps: first, the data is reduced by creating a very small summary whose size has little or no dependency on the original data size; second, the learning algorithm, that remains unchanged, is applied to the summary only, reducing the computing cost in the learning process substantially. The challenge thus resides in balancing the trade-off between the size of the resulting summary and the information we lose in the compression process. We refer to such trade-off as *the compression trade-off* and we will study it in the context of coresets.

3.2. Coresets

Coresets are a powerful algorithmic framework that has been used to analyse big and complex data. In essence, coresets help in quantifying the aforementioned compression trade-off for a specific optimisation problem of interest. A coreset is a small *weighted* set of data *i.e.* a summary, such that the solution found in the summary is provably correct with respect to the solution found in the full data. Ideally, a coreset should be significantly smaller than the original dataset. Furthermore, state-of-the-art coreset results involve coresets whose sizes are *independent* of the original data size (Braverman et al. (2016)).

Most machine learning problems involve defining an optimisation problem to estimate model parameters or to describe other aspects of data. Thus, for a given learning problem, a coreset can be constructed in order to reduce the volume of data. Then we can proceed to learn using the coreset only, discarding the rest of the data, and we will be guaranteed to obtain *approximately* the same result (Feldman et al. (2013)).

3.2.1. Coreset Design

Any algorithm that constructs a coreset for some dataset, can only provide provably correct guarantees for one specific learning problem. Hence, the way we design a coreset construction algorithm entirely depends on the problem we are interested in learning. However, we can still define coresets in the following problem-agnostic fashion.

Definition 1 *(Δ-coreset): Let function f be the objective function of some learning problem and $\mathcal{D}$ the input data. We call $\mathcal{C}$ a Δ-coreset for $\mathcal{D}$ if the following inequality holds:*

$$|f(\mathcal{D}) - f(\mathcal{C})| \leq \Delta |f(\mathcal{D})| \tag{7}$$

where $\Delta > 0$. Inequality (7) is commonly referred as the *coreset guarantee*; and we say that if such guarantee can be obtained by approximating the original dataset $\mathcal{D}$ with a weighted set $\mathcal{C}$ *w.r.t.* function f, then $\mathcal{C}$ is called an Δ-coreset for $\mathcal{D}$.

Notice that Δ establishes a bound for the solution quality found in the coreset. That is, Δ defines *how far* a solution found in the coreset can be from a solution found in the original dataset.

There are three general quantities that need to be bounded when designing a coreset algorithm for a optimisation problem: 1) the size M of the coreset; 2) the error tolerance Δ *i.e.* maximum amount of discrepancy between solutions found in the full data and in the coreset; 3) the running time of the algorithm *i.e.* if constructing the coreset is computationally comparable to solving the problem using the full data, then there is no point in computing a coreset.

Designing a coreset algorithm for a learning problem thus is usually a very challenging task. Furthermore, not every problem allows a practical coreset construction (Munteanu et al. (2018)). Fortunately, there exists a computationally efficient randomised strategy to construct coresets for the LR problem and we will describe it in the next subsections.

3.2.2. Coresets via importance sampling

A naive randomised approach to construct coresets is by doing uniform sampling. We can simply assign probability $1/N$ to each point in $\mathcal{D}$ and then sample according to this distribution. The problem with this oversimplified approach is that we have to sample a very large number of points to ensure relatively low error (Bachem et al. (2017)). This happens because different points make different contributions to the objective function at hand and hence we can easily leave important points out of the coreset if we do not sample sufficiently large coresets. This is why a more sophisticated sampling scheme is desirable.

The most effective randomised approach for constructing coresets is by doing non-uniform sampling, namely *importance sampling* (Feldman and Langberg (2011)). In this approach, rather than just assigning equal probability to all our input data points, we first compute an importance score that tells us "how redundant" a data point is for our learning problem. This score is called the *sensitivity* of the point, and is the central quantity for constructing coresets non-uniformly. Once we computed the sensitivity for each input point, we sample M points according to the sensitivities. The final step is to compute the *weights* for the sampled points, which are generally inverse-proportional to the sensitivity scores, and return the M weighted points.

When constructing coresets using importance sampling, defining and computing the sensitivity is very challenging because (i) computing the exact sensitivity is not computationally tractable, that is, we need to define lower and upper bounds for it and prove that using these bounds yields to small coresets; (ii) the bounds should be efficiently computable. On this front, it is worth mentioning that coresets algorithm need to inspect each input point at least once. Hence, by "efficiently computable" we mean linear time.

One last observation is that results obtained by using importance sampling come in a PAC-like flavour [1]: we have a tuple of error parameters, (Δ,δ), where Δ is the already defined error tolerance and δ is the probability of failure *i.e.* $\delta \in (0, 1)$ and it tells us the chances that after running a randomised coreset algorithm we do not obtain a coreset.

3.2.3. Coreset for Logistic Regression

Having discussed, in general terms, the importance sampling approach for constructing coresets, we now present a state-of-the-art importance sampling algorithm to construct a coreset for the LR problem.

We defined LR in Section 2.1 as the problem we are interested in learning. Therefore, an importance-sampling coreset algorithm will have to identify important points for the log-likelihood function defined in Equation (2). Concretely, the job of such coreset algorithm is to construct a weighted set $\mathcal{C} := \{(w_m, \tilde{x}_m, \tilde{y}_m)\}_{m=1}^{M}$, with $M \ll N$, such that the weighted log-likelihood function $\tilde{\mathcal{L}}_N(\theta|\mathcal{C}) := \sum_{m=1}^{M} w_m \ln p_{logistic}(\tilde{y}_n|\tilde{x}_m;\theta)$ satisfies the coreset guarantee:

$$|\mathcal{L}_N(\theta|\mathcal{D}) - \tilde{\mathcal{L}}_N(\theta|\mathcal{C})| \leq \Delta|\mathcal{L}_N(\theta|\mathcal{D})| \tag{8}$$

for all $\theta \in \mathbb{R}^{d+1}$.

We use the recent coreset construction method proposed by Huggins *et al.* (Huggins et al. (2016)), to which we refer as *One-Shot Coreset Algorithm For Logistic Regression* (OSCA-LR). This coreset construction is guaranteed to produce a Δ-coreset of size M with probability $1 - \delta$ for Logistic Regression. See Algorithm 1 for a description and referenced work for the proof.

OSCA-LR uses importance sampling to select data points with huge contribution to the log-likelihood function (2). This is encapsulated in the procedure **Sensitivity()** in Algorithm 1.

Remark:
Notice that the theoretical description of coreset algorithms, such as the one presented for OSCA-LR in Algorithm 1, requires the error tuple (Δ, δ) as parameters in order to compute the corresponding coreset size M. In practice, however, one does not interact with coreset algorithms via the error parameters; since we already know the number points we can afford to maintain as a coreset *i.e.* given memory, storage, time constraints, etc, we give the number M of desired points to our coreset algorithm and we get a M-sized coreset in return. Thus, we never explicitly set the error parameters as these values are mostly of theoretical interest. We refer the reader to Braverman et al. (2016) for a detailed explanation on the relationship between the errors tuple and the size of the coreset.

1. PAC: Provably Approximately Correct

Algorithm 1: The OSCA-LR algorithm: a coreset algorithm for Logistic Regression, as proposed by Huggins *et al.* D is the number of features in data $\mathcal{D}$ and c is a constant.

Input: Data $\mathcal{D}$, k-clustering $\mathcal{Q}$ with $|\mathcal{Q}| = k$, radius $R > 0$, coreset error $\Delta > 0$, failure parameter $\delta \in (0, 1)$

Output: Δ-coreset $\mathcal{C}$ with probability $1 - \delta$

for $n = 1, 2, ..., N$ **do**

  $m_n \leftarrow$ `Sensitivity(`$N, \mathcal{Q}, R$`)` ; `// Compute the sensitivity of each point`

end

$\bar{m}_N \leftarrow \frac{1}{N} \sum_{n=1}^{N} m_n$

$M \leftarrow \lceil \frac{c\bar{m}_N}{\Delta^2} [(D+1) log\, \bar{m}_N + log(\frac{1}{\delta})] \rceil$; `// Compute coreset size`

for $n = 1, 2, ..., N$ **do**

  $p_n = \frac{m_n}{N\bar{m}_N}$; `// compute importance weight for each point`

end

$(K_1, K_2, ..., K_N) \sim$ `Multi(`M`,`$(p_n)_{n=1}^{N}$`)` ; `// sample coreset points`

for $n = 1, 2, ..., N$ **do**

  $w_n \leftarrow \frac{K_n}{p_n M}$; `// calculate the weight for each coreset point`

end

$\mathcal{C} \leftarrow \{(w_n, x_n, y_n) | w_n > 0\}$

return $\mathcal{C}$

We mentioned in the previous subsection that the sensitivity has to be bounded in order to be computable. OSCA-LR uses a k-clustering of the input data to obtain such a bound. The algorithm also needs a parameter R to compute the sensitivity. The R parameter is a real-valued quantity that bounds the parameter space, $\mathbb{R}^{d+1}$, to an euclidean ball of radius R. We remind the reader that the coreset guarantee in Formula (8) should hold for all parameters $\theta \in \mathbb{R}^{d+1}$. However, Huggins *et al.* showed that a bounded parameter space is needed in to prevent the sensitivity from blowing out to infinity. The proofs and technical details on how/why this clustering-based bound captures points importance for LR can be found in Huggins et al. (2016).

The k-clustering bound brings an intuitive interpretation for the sensitivity: points that are bunched together are redundant while points that are far from other points have more influence over the LR objective function (Reddi et al. (2016), Huggins et al. (2016)). Hence, points far from the centres $\mathcal{Q}$ are assigned high sensitivity scores while close points will get low scores. After computing the sensitivities, the algorithm defines the non-uniform distribution based on these values. Ideally, we want to sample *sensitive* points as often as possible and put them in the coreset. We sample M points *i.e.* using the **Multi()** procedure, we compute their weights, and finally return all the points with non-zero weights. It is worth mentioning that due to its randomised nature, OSCA-LR produces different coresets at each run.

We can see in Figure 1 a single run of OSCA-LR over a toy dataset; we synthetically generated a small amount of points to show how the LR coreset construction works. We can see how the clustering of the data is used to define the sensitivity. Then, we sample according to the sensitivity distribution, compute the weights and obtain a coreset. The only thing left to do in this case is to run a LR solver over the coloured points in plot (d).

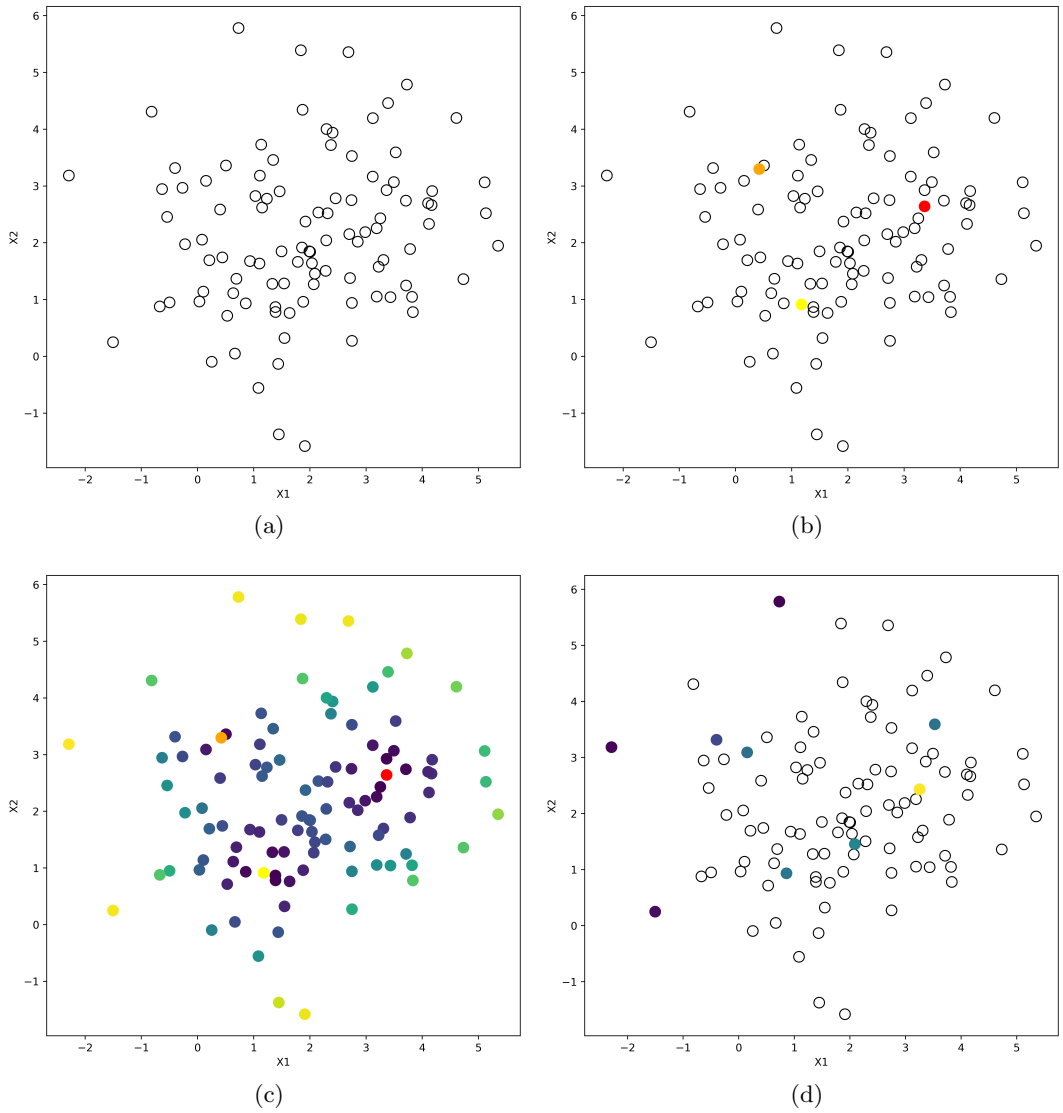


Figure 1: A step-by-step run of OSCA-LR over a simple 2-dimensional synthetic dataset; (a) shows an illustrative synthetic 2-dimensional data for which a logistic regression coreset is computed using OSCA-LR; the data has 100 points; (b) displays a k-clustering of the data to be used by the coreset algorithm to compute the sensitivities, with $k = 3$ (c) shows the sensitivity distribution for our small dataset; the brighter the colour, the more sensitive the point is;the radius was set to $R = 2.5$ (d) shows the obtained coreset, with the coreset size being 9 points; the colours here indicate the weights of points: the brighter the colour, the "heavier" the point is.

Notice that to compute the coreset, we only need the example's feature vector x. The label y is only used in the learning phase [2].

2. Coreset points retain the same labelling.

Important final observations regarding the OSCA-LR algorithm: 1) the inverse relation between the sensitivity and the weight of a point *i.e.* highly sensitive points are low-weighted and vice versa. This is intuitive if we consider the fact that we are using clustering to compute the sensitivity: highly sensitive points are representing few points from the original dataset as they were far from other points. This can be observed in plots (c) and (d) of Figure 1; 2) the compression trade-off between the size of the coreset M and the coreset error Δ: smaller coresets can potentially incur in greater error while a larger coreset can be more accurate at the cost of sacrificing more storage and computing time; 3) the polynomial dependency of the coreset size on the mean sensitivity $\bar{m}$: as the parameter space expands, the mean sensitivity grows, which in turn enlarges the coreset size. Hence, it is clearer now why the parameter space has to be bounded to a ball of radius R *i.e.* unbounded parameter space means infinite sensitivity, which implies infinite coreset size. Again, if we go back one more time to Figure 1, we computed the sensitivities in plot (c) fixing the radius to $R = 2.5$. If we re-run the algorithm with a higher value for R, say $R = 5$, then we would see that the great majority, if not all the points, would turn into a very bright colour. Hence, when the sensitivities are too high, we virtually degenerate into uniform sampling; 4) OSCA-LR can compute all the sensitivity scores in $O(Nk)$ time; we previously mentioned that coresets algorithm cannot do better than linear time as they need to examine each data point at least once. Therefore, OSCA-LR is a quiet efficient coreset algorithm and, as the size of data gets very large, its gains in running time become quite useful.

3.3. Our strategy

We conclude this section with a concise description of how we use the importance-sampling algorithm OSCA-LR to solve a large-scale LR problem with conformal predictors.

Our approach consists in using coresets as a pre-processing step for Conformal Predictors in order to efficiently solve LR. Hence, we divide the computational tasks in two main steps:

1. **compress the input data:** we compute a coreset for our data using the OSCA-LR algorithm. Since OSCA-LR needs a k-clustering of the data, we run the K-Means++ algorithm to obtain the k centres.

2. **learn from the coreset:** after we compress the data, we only keep the small coreset and proceed to solve a weighted instance of LR using inductive conformal predictors. To train the icp model *i.e.* to define the NCM $\mathcal{A}$ from Section 2.2.1, we use the state-of-the-art *LIBLINEAR* solver (Fan et al. (2008)), which has been extensively used to train large-scale linear classification problems.

We refer to the above strategy as *Coreset-Based Conformal Predictors* (CBCPs). Hence, CBCPs learn over compressed data and, as we will see in the next section, this brings an interesting computing-time boost to CPs with negligible validity/efficiency degradation.

4. Experimental results

Having outlined our approach, this section presents our experimental results. We test CBCPs against standalone CPs on real-world datasets. As previously mentioned, we apply

our strategy to the inductive version of CPs since these are the more computationally efficient CPs. In order to remind the reader of this decision, we call our algorithms Inductive Coreset-Based Conformal Predictors (ICBCPs).

With our experiments we answer the following research questions:

- **How do ICBCPs perform computationally when compared to ICPs?** We will show that ICBCPs outperform ICPs when dealing with large datasets.
- **Are validity and efficiency retained with ICBCPs?** We will demonstrate that the efficiency of ICBCPs, in the Conformal Prediction sense, does not exhibit any meaningful change when compared to ICPs. Regarding validity, we will see that in some cases the number of ICBCPs errors is even less than the number of ICP errors, which is an interesting result.

4.1. Experiment description

We evaluated ICBCPs using the datasets detailed in Table 1. All the datasets are public and can be downloaded from https://www.csie.ntu.edu.tw/~cjlin/libsvm/. The Covertype dataset (Blackard and Dean (1999)) contains cartographic features and the labels correspond to different forest cover types; KDD-CUP 2010 (Stamper et al. (2010)) is a massive dataset generated for an educational data mining competition; finally, based on 20000 messages taken from 20 newsgroups, News-20 was generated in Keerthi and DeCoste (2005) for experiments that needed both high data size and dimensionality.

Mind that due to time and hardware constraints, we had to reduce the number of dimensions of the KDD-CUP 2010 and News20 datasets. Since our focus resides entirely in measuring computing time and CP properties, not in improving prediction accuracy, we solved this issue by simply taking the first 2,000 features for KDD-CUP and the first 50,000 features for News-20.

We conducted our experiments as follows: we fix 5 algorithms for testing, 4 of which are ICBCPs using different coreset sizes and the remaining one is the standalone ICP. Concretely, our algorithms are $\text{ICBCP}_{0.1\%}$, $\text{ICBCP}_{0.5\%}$, $\text{ICBCP}_{1\%}$, $\text{ICBCP}_{5\%}$ and ICP, where $\text{ICBCP}_{g\%}$ means that we set the coreset size M to be g percent of the (proper) training set size N. Notice that this means we construct our coresets using M, the size of the desired coreset, as a parameter *i.e.* we do not specify the tuple (Δ,δ), as discussed in Section 3.2.3. For each dataset in Table 1 and for each of the above algorithms, we do the following computations:

1. **shuffle all the data:** some datasets, like Covertype and News20, are packed in only one big file. For these datasets, we fully shuffled the data. However, competition datasets like KDD-CUP 2010 specify in different files the portion of the data to use for training and testing, respectively. For this case, we shuffled only the training data, *i.e.* we did not mix training and testing examples.
2. **split the data:** we split the samples as described in Section 2.2.1, that is, we split it into the proper training set, which will be used to train a model, the calibration set, which will be used to calibrate this model, and the test set for testing this model.

3. **compute a model:** here is where ICP differs from the other algorithms. For ICP we proceed as usual: we train a LR model over the proper training set and use the obtained model to assign nonconformity scores to the calibration set examples. For ICBCPs, however, we cluster the proper training set using the K-Means++ algorithm. We used a value of $k = 4$ for K-Means++ and a radius of $R = 1$ for OSCA-LR, for all our ICBCPs algorithms. Next, we feed the OSCA-LR algorithm with the proper training set and its clustering to obtain a coreset. Finally, we use the coreset to solve the LR problem with non-uniform weights.

4. **make predictions and evaluate performance:** we finally make the prediction for the test instances and apply our performance metrics, described in Section 4.2.

Table 1: The 3 datasets used in our experiments are Covertype, News20 and KDD-CUP 2010. For all them, we used 60% of the data for the proper training set, 15% for the calibration set and 25% for the test set.

Dataset	Examples	Features
News20	19,996	1,355,191
Covertype	581,012	54
KDD-CUP 2010	8,918,054	20,216,830

We computed the above procedure 20 times for each of the algorithms. Regarding the hardware, our experiments were performed on a single desktop PC running Ubuntu-Linux operating system, equipped with an Intel(R) Xeon(R) CPU E3-1225 v5 @ 3.30GHz processor and 32 Gigabytes of RAM.

Lastly, for coresets, we adapted to our needs the OSCA-LR algorithm implemented and shared by its authors [3]. For Conformal Prediction, we used the Nonconformist library[4]. All our programs were written using the Python programming language.

4.2. Evaluation criteria

To evaluate our results, we set out the following criteria, which are most relevant in both the coreset and Conformal Prediction domains.

- **Computing time:** our main criterion is the time, in seconds, that is needed for each of the five algorithms to compute prediction sets for each instance in the test set. We do not take into account the data splitting time nor the time needed to compute our metrics and store the results. Even though the coreset algorithm OSCA-LR assumes that the clustering of the data is given as an input, this assumption is non-realistic. Hence, for the coreset-based predictors, we measure the clustering time and the proper coreset construction time.

3. https://bitbucket.org/jhhuggins/lrcoresets/src/master - last accessed in 5/2019
4. https://github.com/donlnz/nonconformist - last accessed in 5/2019

- **Validity:** we measure the validity of each of the considered algorithms. That is, we count the number of prediction errors made by all five algorithms, in the conformal-prediction sense.
- **Efficiency:** we apply four efficiency measures from Vovk et al. (2016); namely, the average prediction size, the p-values mean, the rate of singleton prediction and the rate of empty predictions.

4.3. Computing time performance

We now present our results measuring the computing time of our five algorithms. Notice that the small size of the coresets was intended to verify if we can still maintain the same efficiency and validity of ICP. This will be examined in the next subsection.

Again, the computing time was measured as described in Section 4.2 and the experiment was conducted as detailed in Section 4.1.

Figure 2 illustrates the significant benefit of ICBCP, which consistently outrun standard ICP, across all three datasets. In particular, we observed nearly 5 times faster running time for the KDD-CUP 2010 dataset, and 2 times faster for the News20 dataset. It is worth pointing out that larger coreset size does not always result in longer running time, as the non-deterministic clustering convergence time may vary depending on the structure of the dataset.

Interestingly, using only a very small fraction of the proper training samples, like 0.1% or 0.5%, ICBCPs not necessarily reduce the prediction accuracy of ICP. In fact, we will complete our results exposition in the next subsection by showing that validity and efficiency might even improve with coresets.

4.4. Efficiency and validity of predictions

We showed that coreset-based conformal prediction can be a major improvement in terms of computing time for inductive conformal prediction. We now examine in detail how validity and efficiency behave under the presence of compressed data.

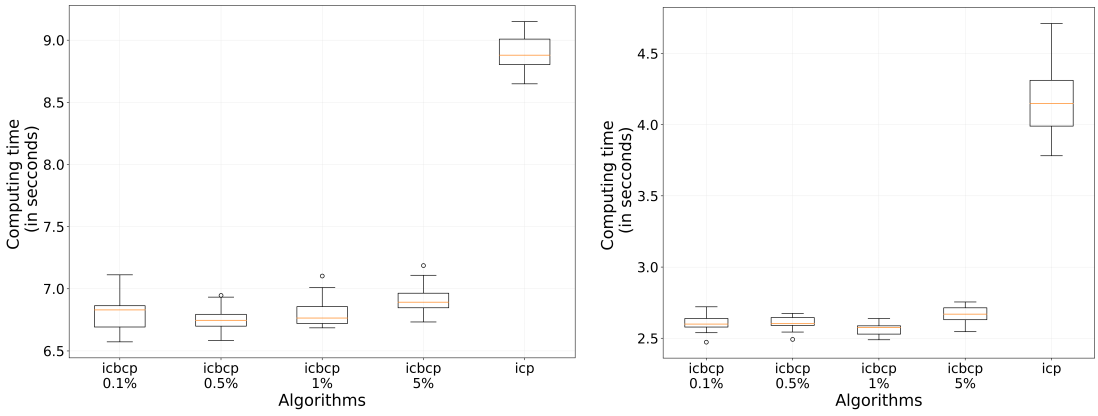
Once more, we followed the same experiment procedure described previously.

To assess the validity of ICBCP, we take a closer look at the error rates across all 100 confidence levels, averaged over 20 runs with all test samples. Figure 3 suggests that our approach retained full validity for all confidence levels, with the majority of the error rates around the perfect diagonal calibration line, subjected to statistical fluctuations.

To gain deeper insight into the effect of ICBCP, we inspect the number of empty prediction sets, and the number of single prediction set. In essence, the higher the significance level, the more empty prediction sets we have (see Figure 4).

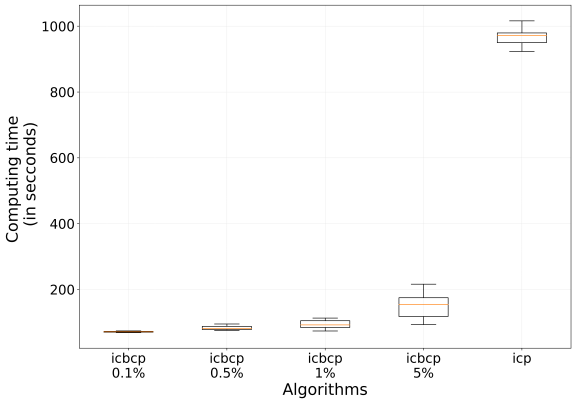
5. Related work on coresets

The technique of coresets was born in the field of computational geometry as a systematic approach for approximating the optimal solution for *shape-fitting problems* such as the *Minimum Enclosing Ball* (MEB) Agarwal and Sharathkumar (2010) Badoiu and Clarkson (2003) Bădoiu and Clarkson (2008) Munteanu et al. (2014) and *Directional Width* (DW) Agarwal et al. (2004) Chan (2006). In machine learning, coresets were mainly studied in the



(a) Covertype dataset

(b) News20 dataset



(c) KDD-CUP 2010 dataset

Figure 2: Comparison of the computing time between ICBCP and ICP on 3 datasets. The central line in each box indicates the median. The bottom and top of the box indicate the 25^{th} and 75^{th} percentiles respectively. The small circles denote the outliers. ICBCP consistently outruns ICP on all three datasets. Different coreset sizes does not seem to meaningfully impact on the computing time of ICBCP.

context of unsupervised learning Feldman et al. (2013) Har-Peled and Mazumdar (2004) Bachem et al. (2017) Zhang et al. (2017) Ackermann et al. (2012) to devise fast approximation and streaming algorithms for computationally-intractable problems *e.g.* clustering.

There were significantly less incursions in the supervised-learning area. Reddi *et al.* Reddi et al. (2015) explored coresets for the problem of *Empirical Risk Minimisation*, central in statistical learning theory. Specifically, for the Logistic and Hinge loss functions, they proposed an iterative gradient-based coreset algorithm by exploiting the following observation: at each iteration of the optimisation process, only a small set of points are important *i.e.* those that contribute the most to the objective function. Thus, they showed that the problem can be solved over this compact set of points, obtaining a provable good solution. One interesting observation made by Reddi *et al.* is that the process of computing the

Table 2: The performance of ICP and ICBCP on the Covertype dataset, averaged over 20 runs, with 145,253 test samples. icbcp-0.1%, icbcp-0.5%, icbcp-1% and icbcp-5% denote the size of the coresets corresponding to 0.1%, 0.5%, 1% and 5% of the original dataset size, respectively. Lower prediction size and lower error rate are generally desirable. Overall, our ICBCP closely matches ICP across all criteria.

(a) 99% confidence level, $\epsilon = 0.01$

Algorithm	**Average prediction size**	**Mean p-values**	**Single prediction (%)**	**Empty prediction (%)**	**Error rate**
icp	1.916	0.558	0.028	0.028	0.0299
icbcp-0.1%	1.936	0.517	0.025	0.0197	0.0303
icbcp-0.5%	1.944	0.536	0.021	0.0177	0.0299
icbcp-1%	1.942	0.54	0.019	0.019	0.03
icbcp-5%	1.929	0.553	0.018	0.026	0.0299

(b) 95% confidence level, $\epsilon = 0.05$

Algorithm	**Average prediction size**	**Mean p-values**	**Single prediction (%)**	**Empty prediction (%)**	**Error rate**
icp	1.876	0.557	0.042	0.041	0.0499
icbcp-0.1%	1.893	0.555	0.023	0.042	0.0499
icbcp-0.5%	1.883	0.557	0.037	0.04	0.05
icbcp-1%	1.883	0.558	0.035	0.041	0.05
icbcp-5%	1.878	0.558	0.042	0.04	0.0499

(c) 70% confidence level, $\epsilon = 0.3$

Algorithm	**Average prediction size**	**Mean p-values**	**Single prediction (%)**	**Empty prediction (%)**	**Error rate**
icp	1.473	0.558	0.268	0.1297	0.3
icbcp-0.1%	1.423	0.517	0.212	0.182	0.3
icbcp-0.5%	1.455	0.536	0.183	0.181	0.299
icbcp-1%	1.454	0.54	0.19	0.178	0.3
icbcp-5%	1.469	0.553	0.256	0.138	0.3

well-known gradient descent algorithm can be seen as an instantiation of their framework where the coreset consists in the gradients computed at each iteration

Huggins *et al.* (Huggins et al. (2016)) proposed a coreset algorithm assuming very similar supervised-learning settings but from a Bayesian perspective *i.e.* Bayesian Logistic Regression. They proved that, by approximating (up to a multiplicative factor) the log-likelihood of the observations using a weighted subset of the data, one can obtain a coreset with high probability.

The work of Huggins *et al.* was our main motivation for using coresets as a pre-processing step for conformal prediction due to their good results with computationally expensive

Table 3: The performance of ICP and ICBCP on the KDD-CUP 2010 dataset, averaged over 20 runs, with 510,302 test samples. Lower prediction size and lower error rate are generally desirable. Overall, our ICBCP closely matches ICP across all criteria.

(a) 99% confidence level, $\epsilon = 0.01$

Algorithm	**Average prediction size**	**Mean p-values**	**Single prediction (%)**	**Empty prediction (%)**	**Error rate**
icp	1.523	0.307	0.477	0	0.033
icbcp-0.1%	1.741	0.333	0.259	0	0.033
icbcp-0.5%	1.674	0.321	0.326	0	0.028
icbcp-1%	1.609	0.313	0.391	0	0.03
icbcp-5%	1.677	0.318	0.323	0	0.026

(b) 95% confidence level, $\epsilon = 0.05$

Algorithm	**Average prediction size**	**Mean p-values**	**Single prediction (%)**	**Empty prediction (%)**	**Error rate**
icp	1.363	0.307	0.637	0	0.05
icbcp-0.1%	1.586	0.333	0.414	0	0.054
icbcp-0.5%	1.531	0.321	0.469	0	0.043
icbcp-1%	1.467	0.313	0.533	0	0.047
icbcp-5%	1.507	0.318	0.493	0	0.043

(c) 70% confidence level, $\epsilon = 0.3$

Algorithm	**Average prediction size**	**Mean p-values**	**Single prediction (%)**	**Empty prediction (%)**	**Error rate**
icp	0.755	0.307	0.755	0.245	0.312
icbcp-0.1%	0.818	0.333	0.818	0.182	0.296
icbcp-0.5%	0.836	0.321	0.821	0.172	0.278
icbcp-1%	0.817	0.313	0.811	0.186	0.278
icbcp-5%	0.86	0.318	0.79	0.175	0.271

Bayesian-inference algorithms. In a sense, we also indirectly complemented their work by using their method in the optimisation setting *i.e.* non-Bayesian, where they did not test the OSCA-LR algorithm, and of course, by bringing the technique to conformal prediction.

6. Conclusion and future work

Many machine learning algorithms are computationally expensive, making their direct applications to large-scale datasets difficult or infeasible. In this paper, we have proposed Coreset-Based Conformal Prediction, a strategy for dealing with large-scale data within the conformal prediction framework. To the best of our knowledge, our work is the first attempt to use coresets along with CP. Coreset-based conformal predictors can use a small fraction

Table 4: The performance of ICP and ICBCP on the News20 dataset, averaged over 20 runs, with 4,999 test samples. icbcp-0.1%, icbcp-0.5%, icbcp-1%, icbcp-5% denote the size of the coresets corresponding to 0.1%, 0.5%, 1% and 5% of the original dataset respectively. Lower prediction size and lower error rate are generally desirable. Overall, our ICBCP closely matches ICP across all criteria.

(a) 99% confidence level, $\epsilon = 0.01$

Algorithm	**Average prediction size**	**Mean p-values**	**Single prediction (%)**	**Empty prediction (%)**	**Error rate**
icp	1.941	0.5	1.00020004e-05	0.029	0.029
icbcp-0.1%	1.941	0.5	2.00040008e-05	0.0295	0.0295
icbcp-0.5%	1.941	0.5	3.00060012e-05	0.029	0.0293
icbcp-1%	1.94	0.5	0	0.0301	0.0301
icbcp-5%	1.941	0.5	1.00020004e-05	0.029	0.029

(b) 95% confidence level, $\epsilon = 0.05$

Algorithm	**Average prediction size**	**Mean p-values**	**Single prediction (%)**	**Empty prediction (%)**	**Error rate**
icp	1.9	0.5	4.00080016e-05	0.0498	0.0499
icbcp-0.1%	1.9	0.501	3.00060012e-05	0.0499	0.0499
icbcp-0.5%	1.9	0.501	1.00020004e-05	0.049	0.049
icbcp-1%	1.9	0.501	2.00040008e-05	0.0498	0.0498
icbcp-5%	1.901	0.5	7.00140028e-05	0.049	0.049

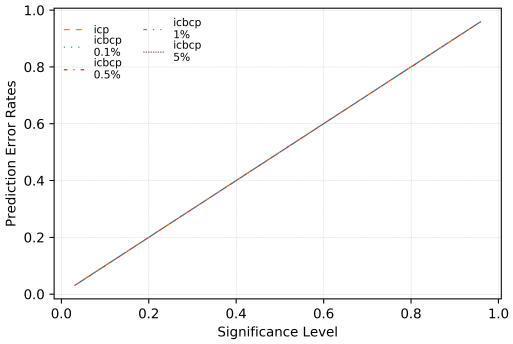
(c) 70% confidence level, $\epsilon = 0.3$

Algorithm	**Average prediction size**	**Mean p-values**	**Single prediction (%)**	**Empty prediction (%)**	**Error rate**
icp	1.399	0.5	0.00016	0.3	0.3
icbcp-0.1%	1.4	0.501	0.00024	0.2996	0.2997
icbcp-0.5%	1.4	0.501	0.00016	0.299	0.2996
icbcp-1%	1.407	0.501	0.00016	0.296	0.296
icbcp-5%	1.4	0.5	0.00018	0.2997	0.2997

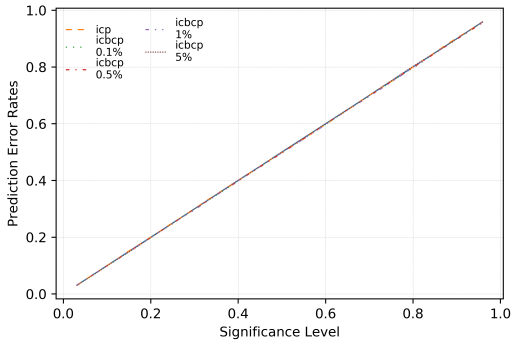
of the original dataset, namely the coreset, and learn using conformal predictors. We have tested our approach with different coreset sizes over 3 public datasets to demonstrate that our proposed CBCP may run up to 5 times faster than the ICP on the full dataset, while achieving the similar prediction accuracy and Conformal Prediction's validity.

We also observed that once the data is large, the time it takes to build the coreset is small compared to the time needed to learn over the full dataset.

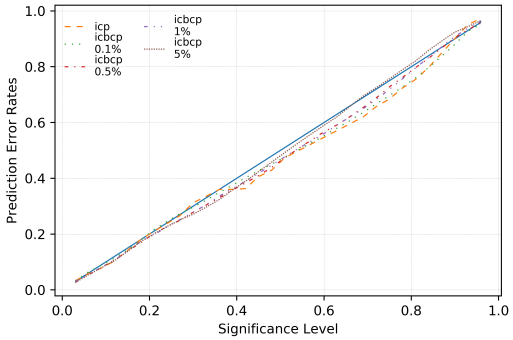
One of possible future research directions is to extend our approach to the streaming setting. In particular, coresets proved to be very useful in scenarios where there is limited storage, no random access to the data and massive amount of incoming information that



(a) Covertype dataset, with 145,253 test samples



(b) News20 dataset, with 4,999 test samples



(c) KDD-CUP 2010 dataset, with 510,302 test samples

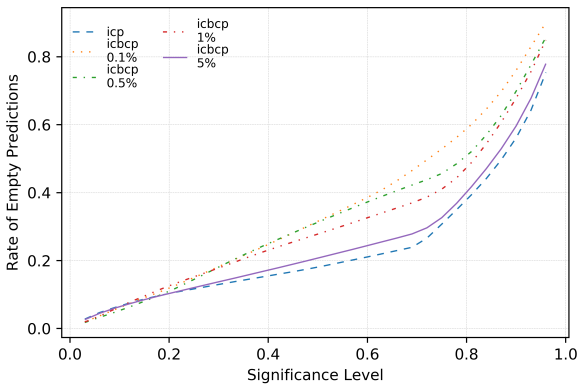
Figure 3: The validity of ICBCP on 3 datasets, averaged over 20 runs. All 100 confidence levels were assessed to observe that ICBCP produced valid predictions, subject to statistical fluctuations. In each plot, the straight blue line corresponds to the "perfect" calibration line.

have to be used for solving some specific problem. It is stimulating to see that both coresets and conformal predictors have seen their best results in the online model.

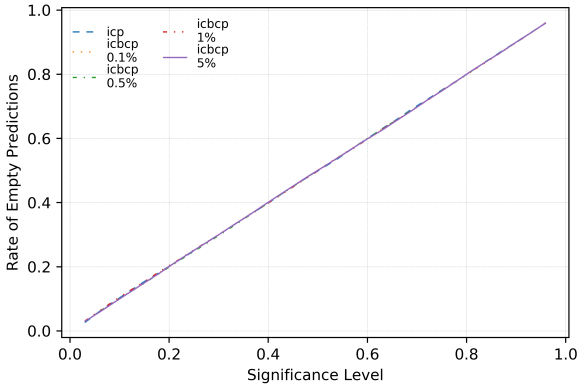
Finally, we are also interested in studying CBCP in distributed environments, where the data is inherently distributed and the participating machines have to collaboratively learn a model efficiently.

Acknowledgments

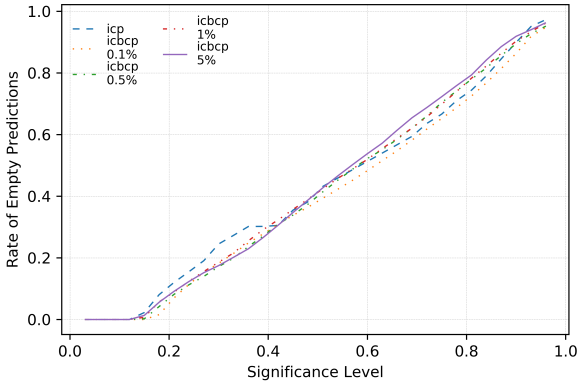
We thank the COPA-2019 reviewers for their thoughtful and thorough reviews. We also thank Astra Zeneca and the Paraguayan Government for their support.



(a) Covertype dataset, with 145,253 test samples



(b) News20 dataset, with 4,999 test samples



(c) KDD-CUP 2010 dataset, with 510,302 test samples

Figure 4: Comparison of the rate of empty prediction sets produced by ICBCP and ICP, on 3 datasets, averaged over 20 runs. Our proposed ICBCP closely matches ICP in all test cases.

References

Marcel R Ackermann, Marcus Märtens, Christoph Raupach, Kamil Swierkot, Christiane Lammersen, and Christian Sohler. Streamkm++: A clustering algorithm for data streams. *Journal of Experimental Algorithmics (JEA)*, 17:2–4, 2012.

Pankaj K Agarwal and R Sharathkumar. Streaming algorithms for extent problems in high dimensions. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete algorithms*, pages 1481–1489. SIAM, 2010.

Pankaj K Agarwal, Sariel Har-Peled, and Kasturi R Varadarajan. Approximating extent measures of points. *Journal of the ACM (JACM)*, 51(4):606–635, 2004.

Olivier Bachem, Mario Lucic, and Andreas Krause. Practical coreset constructions for machine learning. *arXiv preprint arXiv:1703.06476*, 2017.

Mihai Badoiu and Kenneth L Clarkson. Smaller core-sets for balls. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 801–802. Society for Industrial and Applied Mathematics, 2003.

Mihai Bădoiu and Kenneth L Clarkson. Optimal core-sets for balls. *Computational Geometry*, 40(1):14–22, 2008.

Vineeth Balasubramanian, Shen-Shyang Ho, and Vladimir Vovk. *Conformal prediction for reliable machine learning: Theory, Adaptations and applications*. Newnes, 2014.

J A Blackard and D J Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and Electronics in Agriculture*, vol.24:131–151, 1999.

Vladimir Braverman, Dan Feldman, and Harry Lang. New frameworks for offline and streaming coreset constructions. *CoRR*, abs/1612.00889, 2016. URL http://arxiv.org/abs/1612.00889.

Timothy M Chan. Faster core-set constructions and data-stream algorithms in fixed dimensions. *Computational Geometry*, 35(1-2):20–35, 2006.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *Journal of machine learning research*, 9 (Aug):1871–1874, 2008.

Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 569–578. ACM, 2011.

Dan Feldman, Melanie Schmidt, and Christian Sohler. Turning big data into tiny data: Constant-size coresets for k-means, pca and projective clustering. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 1434–1453. SIAM, 2013.

Alexander Gammerman and Vladimir Vovk. Hedging predictions in machine learning: The second computer journal lecture. *The Computer Journal*, 50(2):151–163, 2007.

Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 291–300. ACM, 2004.

Jonathan Huggins, Trevor Campbell, and Tamara Broderick. Coresets for scalable bayesian logistic regression. In *Advances in Neural Information Processing Systems*, pages 4080–4088, 2016.

S Sathiya Keerthi and Dennis DeCoste. A modified finite newton method for fast solution of large scale linear svms. *Journal of Machine Learning Research*, 6(Mar):341–361, 2005.

Jonathan Milgram, Mohamed Cheriet, and Robert Sabourin. one against one or one against all: Which one is better for handwriting recognition with svms? In *Tenth international workshop on frontiers in handwriting recognition*. Suvisoft, 2006.

Alexander Munteanu, Christian Sohler, and Dan Feldman. Smallest enclosing ball for probabilistic data. In *Proceedings of the thirtieth annual symposium on Computational geometry*, page 214. ACM, 2014.

Alexander Munteanu, Chris Schwiegelshohn, Christian Sohler, and David Woodruff. On coresets for logistic regression. In *Advances in Neural Information Processing Systems*, pages 6561–6570, 2018.

Jeff M Phillips. Coresets and sketches. *arXiv preprint arXiv:1601.00617*, 2016.

Sashank J Reddi, Barnabás Póczos, and Alexander J Smola. Communication efficient coresets for empirical loss minimization. In *UAI*, pages 752–761, 2015.

Sashank J Reddi, Jakub Konečnỳ, Peter Richtárik, Barnabás Póczós, and Alex Smola. Aide: fast and communication efficient distributed optimization. *arXiv preprint arXiv:1608.06879*, 2016.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

J Stamper, A Niculescu-Mizil, S Ritter, GJ Gordon, and KR Koedinger. Algebra i 2008–2009. challenge data set from kdd cup 2010 educational data mining challenge. *Retrieved April*, 25:2015, 2010.

Vladimir Vapnik. *Statistical learning theory. 1998*, volume 3. Wiley, New York, 1998.

Vladimir Vovk. Conditional validity of inductive conformal predictors. In *Asian conference on machine learning*, pages 475–490, 2012.

Vladimir Vovk, Alex Gammerman, and Glenn Shafer. *Algorithmic learning in a random world*. Springer Science & Business Media, 2005.

Vladimir Vovk, Valentina Fedorova, Ilia Nouretdinov, and Alexander Gammerman. Criteria of efficiency for conformal prediction. In *Symposium on Conformal and Probabilistic Prediction with Applications*, pages 23–39. Springer, 2016.

Yu Zhang, Kanat Tangwongsan, and Srikanta Tirthapura. Streaming k-means clustering with fast queries. In *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*, pages 449–460. IEEE, 2017.