Coreset-based Protocols for Machine Learning Classification

Nery Anibal Riquelme-Granada
Department of Computer Science
Royal Holloway, University of London

A thesis submitted for the degree of
Doctor of Philosophy

Surrey 2021

This thesis is dedicated to my grandfather Juan Evangelista, with whom I coincided on Earth only for five days, and yet his memory endlessly carries on within my heart.

...

Esta tesis está dedicada a mi abuelo Juan Evangelista. Solo tuvimos 5 días para compartir juntos; y hoy, más de 30 años después, su presencia y recuerdos siguen intactos en mi corazón.

Declaration of Authorship

I, Nery Aníbal Riquelme-Granada, hereby declare that this thesis and the work presented in it is entirely my own. Where I have consulted the work of others, this is always clearly stated.

Signed:

Date:

Acknowledgements

This journey, without the shadow of a doubt, has been the most transcendental one in my life; far from being reducible only to its academic aspect, it has touched, and I would even dare writing *improved*, virtually every dimension of my persona. Indeed, reaching this point of the adventure would not have been possible had I not been supported by a wonderful group of people. First, I thank God, who I believe is *Ipsum Esse* - the sheer act of *to be* itself, for I have witnessed a beautiful glimpse of his presence, his heart and his mind, like never before, during these last years. From the noisy halls of the university in the autumn term, to the peaceful quiet of my home, I know He was there all along. I thank my beautiful wife, Natalia, who provided me with unbounded amount of motivation, determination, and above all, love, even when that meant a delay in her professional career. We laughed together, cried together, fought together, stressed together...we did all of it, together. I thank my parents, Nery and Myriam, who have taught me to pursue knowledge and truth since I was little, always within the beautiful confines of humility and modesty. I vividly recall that one night when, after having dinner, they told us they worked day and night, not to leave their children any material assets (the grocery store they own was not doing well), but to put in our hands a university diploma, which they claimed "is one of the finest weapons one can possess". I carry their teachings always in my heart. I thank my three sisters: Micham, Susy and Fatima, for all those beautiful memories we shared together; those memories were lethal weapons against the feeling of isolation, which, unfortunately, became all too common for almost two years now. I thank my supervisors, Zhiyuan Luo and Khuong Nguyen, for being there for me when I needed them the most, helping me with diligence and patience, always with a friendly smile, especially when the energies were dangerously low. They knew how to talk to me when I needed someone to talk to, they knew how to push me when my motivation was lacking, they knew how to stop me when my curious mind would impulsively go for the branches, and they knew how to teach me to be precise, careful and detail-oriented. I thank the staff from the Computer Science department at Royal Holloway for providing me with all the resources necessary for undertaking my research, for making me feel at home. I thank the Paraguayan National Scholarship Program for Graduate Studies Abroad "Don Carlos Antonio López" for funding my research and my life in the UK; without them

none of this would have been possible. Last but not least, I thank my friends and colleagues for supporting me in different forms: a cup of coffee, research discussions, an unexpected text message, lunch together, discord calls, good and bad jokes, diner together, jam sessions, a sincere smile... I will never forget those gestures.

I give to all these people my eternal and most sincere gratitude.

Agradecimientos

Esta travesía ha sido, sin lugar a dudas, la aventura más trascendental que me ha tocado vivir hasta el día de hoy. Se me hace difícil poder aislar la experiencia de realizar un doctorado tan solo al ámbito académico, pues la misma ha tocado tantas partes de mi vida que hasta me animaría a decir que el proceso me ha ayudado a mejorar cada parte de mi persona. No es exageración decir que este logro no hubiera sido posible sin el apoyo y el calor humano de maravillosas personas. En primer lugar, doy las gracias a Dios, Quien firmemente he llegado a creer es *Ipsum Esse* - el pleno acto de la propia existencia - y Quien me acompañó durante todos estos años con hermosos y misteriosos destellos de su mente, su corazón, su voluntad y su amor. Desde los bulliciosos pasillos en la universidad en épocas de clases, hasta la silenciosa y acogedora quietud de mi hogar, sé que Él estuvo presente en todo momento. Agradezco a mi maravillosa esposa, Natalia, quien me suministró con un sinfín de palabras, gestos y actos de motivación, ternura y amor; eligiendo sacrificar proyectos personales para acompañarme en ésta aventura. Pasamos por tantas cosas juntos: risas, alegrías y tristezas... siempre, pero siempre, juntos. Agradezco a Papá y Mamá, Nery y Myriam, por haberme impartido el deseo de elegir el camino del conocimiento y la verdad desde que era pequeño, siempre dentro de los bellos confines de la humildad y la modestia. Llevo en el fondo del corazón el recuerdo de aquella noche en la que, después de la cena en familia, nos dijeron que ellos no tendrían los recursos para dejarles a sus hijos una gran herencia o bienes materiales, sino un diploma universitario en mano; un diploma que, ellos dicen, es una de las mejores armas que alguien puede tener para enfrentar la vida. Las enseñanzas de mis padres siempre quedarán en un lugar especial del corazón. Agradezco a mis tres hermanas por todos los hermosos recuerdos y momentos que me regalaron y siguen regalando hasta el día de hoy. Esos recuerdos son fuertes antídotos cuando la nostalgia, el aislamiento y el "techaga'u" atacan. Agradezco a mis supervisores: Zhiyuan Luo y Khuong Nguyen, quienes me acogieron cuando más lo necesité, ofreciendo su valiosa ayuda, siempre con paciencia, tranquilidad y con una amigable sonrisa, especialmente en los momentos en los que las energías se desplomaban peligrosamente; supieron cómo hablarme, escucharme, motivarme, guiarme y enseñarme a hacer investigación con delicadeza y precisión. Agradezco a todo el personal académico y administrativo del Departamento de Ciencias de la

Computación de la Royal Holloway por haberme dado, sin escatimar, los recursos necesarios para ejecutar mi investigación y por haberme hecho sentir como en casa. Agradezco al Programa Nacional de Becas de Paraguay “Don Carlos Antonio López” (BECAL) por brindarme la oportunidad de vivir y estudiar en el Reino Unido, cosa que sin su apoyo hubiese sido imposible.

Finalmente, doy gracias a amigos, familiares y colegas por haberme demostrado su apoyo y cariño en tantas formas y colores: una taza de café, un almuerzo, charlas académicas, mensajes y llamadas inesperadas, llamadas en discord, bromas buenas y malas, una cena juntos, noches de peñas, una sonrisa honesta... de verdad nunca voy a olvidar éstos detalles. Todas estas personas maravillosas, reciban mi sincero y eterno agradecimiento.

Abstract

This thesis addresses the question whether algorithms from Reliable Machine Learning (RML), a family of machine learning frameworks that allow for theoretically bounded prediction errors, can be accelerated in order to allow their use over large data-sets. In particular, this investigation focuses on two fundamental members of the RML family: Conformal Prediction (CP) and Venn-Abers Predictors (VAP). The former consists in a machine learning framework that complements the output of traditional supervised-learning algorithms with reliability measures to model the uncertainty in the predictions. The latter allows for non-probabilistic classifiers to produce well-calibrated probabilities over all possible classes. The additional information that these methods generate, however, come at the price of non-negligible computational overhead. Current state-of-the-art approaches propose the use of these methods in an *inductive setting*, which indeed reduces their computing cost meaningfully. Still, the use of these methods remains restricted to relatively small data-sets. This thesis proposes the acceleration of CP and VAP by using the idea of data summarisation. Specifically, we design methods that rely on reducing the available input data into a coreset: a representation of the input data which retains its main properties while being orders of magnitude smaller than the original data-set. This idea is formalised with two contributions: Coreset-based Inductive Conformal Prediction (C-ICP) and Inductive Venn-Abers Predictors with Enclosing Balls (IVAP-WEB). The obtained results show that both methods indeed give a substantial speed-up to CP and VAP, while largely preserving their predictive performance. This work's third and fourth original contributions are protocols that accelerate the computation of certain types of coresets: Accelerated Clustering via Sampling (ACvS) and Regressed Data Summarisation Framework (RDSF). The numerical results obtained indicate that both ACvS and RDSF efficiently produce high-quality summaries of data, and hence these methods can constitute important tools alongside coresets.

This investigation studies the above frameworks and protocols under the lenses of two fundamental machine learning methods: Logistic Regression and Support Vector Machines, both well-studied at the RML and coreset communities. The proposed methodologies constitute, to the best of our knowledge, the first incursion in using coresets alongside CP and VAP; and, we believe, they open the door

for the study of further interactions between data summarisation techniques and other members of the RML family, such as Venn Machines, Mondrian Prediction and Conformal Predictive Distributions.

List of PhD Publications

1. Riquelme-Granada, N., Nguyen, K., & Luo, Z. (2019, August). Coreset-based conformal prediction for large-scale learning. In Conformal and Probabilistic Prediction and Applications (pp. 142-162). PMLR vol. 105.

2. Riquelme-Granada, N., Nguyen, K. A., & Luo, Z. (2020, July). On Generating Efficient Data Summaries for Logistic Regression: A Coreset-based Approach. In 9th International Conference on Data Science, Technology and Applications (DATA 2020) (pp. 78-89).
   (Best Student Paper Award)

3. Riquelme-Granada, N., Nguyen, K. A., & Luo, Z. (2020, August). Fast probabilistic prediction for kernel SVM via enclosing balls. In Conformal and Probabilistic Prediction and Applications (pp. 189-208). PMLR vol. 128.

4. Riquelme-Granada N., Nguyen K.A., Luo Z. (2021) Coreset-Based Data Compression for Logistic Regression. In: Hammoudi S., Quix C., Bernardino J. (eds) Data Management Technologies and Applications. DATA 2020. Communications in Computer and Information Science, vol 1446. Springer, Cham. https://doi.org/10.1007/978-3-030-83014-4_10

Contents

List of Figures

List of Tables

List of Abbreviations

AA	Approximation Algorithm
ACvS	Accelerated Clustering via Sampling
AVM	Approximation Vector Machines
CP	Conformal Prediction
C-ICP	Coreset-based Inductive Conformal Prediction
ICP	Inductive Conformal Prediction
IID	Independently and Identically Distributed
IVAP	Inductive Venn-Abers Prediction
IVAP-WEB	Inductive Venn-Abers Prediction with Enclosing Balls
LR	Logistic Regression
ML	Machine Learning
NFL	No Free Lunch Theorem
OLS	Ordinary Least Squares
RCA	Randomised Coreset Algorithm
RDSF	Regressed Data Summarisation Framework
RML	Reliable Machine Learning
SVM	Support Vector Machines
VAP	Venn-Abers Prediction

List of Symbols

$\mathbb{R}$	set of real numbers
$\mathbb{R}^+$	set of positive real numbers
$\mathbb{N}$	set of natural numbers
$\mathbb{R}^d$	set of d-dimensional vectors (Euclidean Space)
$a = b$	a is equal to b
$a := b$	a is by definition equal to b
$x \cdot z$	dot product between vector x and vector z
$(0, 1)$	open interval between 0 and 1
$\Theta \subseteq \mathbb{R}^{d+1}$	learning model space (a set)
$\theta \in \Theta$	learning model (a vector)
Z^n	a bag, or multi-set, of n elements
Z^*	set of all bags containing n elements
$\epsilon \in (0, 1)$	significance parameter for CP
$\mathcal{A} : Z^* \times Z \to \mathbb{R}$	non-conformity measure (a function)
$\Gamma : Z^* \times Z \times (0, 1) \to 2^Y$	a Conformal Predictor (a function)
p_y	p-value computed by CP for label y
$\alpha \in \mathbb{R}^+$	non-conformity score
$\mathcal{C} \subseteq \mathbb{R}^d$	a coreset
$X \subseteq \mathbb{R}^d$	object space (a set)
$Y \subseteq \{-1, 1\}$	label space (a set)
$Z \subseteq X \times Y$	example space (a set)
$\mathcal{D} \subseteq Z$	input data (a set)
$x \in X$	an input object (a vector)
$x' \in \mathbb{R}^{d+1}$	an input object padded with 1
$y \in Y$	label for an input object (a scalar)

$z \in Z$	an input example (a tuple)
$n \in \mathbb{N}$	size of the input data
$M \in \mathbb{N}$	coreset size
$\Delta \in \mathbb{R}^+$	coreset error parameter
$k \in \mathbb{N}$	number of centres for k-Means clustering
$h : X \to Y$	a ML hypothesis or classifier (a function)
ln	natural logarithm

Either mathematics is too big for the human mind
. . . or the human mind is more than a machine.

— Kurt Gödel
Essays on Gödel's Reception of Leibniz, Husserl, and Brouwer. ([1])

1 Introduction

This introductory chapter aims to motivate the problems that this thesis covers, and to outline solutions for solving them. In the same measure, it will help the reader place this work and its contributions within the current impressively huge picture of Machine Learning and Artificial Intelligence. The set of research questions that nurtured this work will be presented once the computational scenario is properly defined. Finally, a short description for each of the remaining chapters will be provided.

1.1 Machine Learning

It is instructive to start our exposition with the very definition of the field we are placing ourselves in. Machine Learning (ML) can be defined as the field concerned with automated detection of meaningful patterns, carried out through computer programs. The word "automated" pinpoints the crucial difference from traditional computer programs, where a human programmer writes explicit fine-detailed instructions for each task that the program will, or will not, handle. A machine learning program, however, deals with input data containing patterns that are so complex that it is not feasible for the programmer to write explicit instructions for handling them. A key characteristic of machine learning programs is the use of

the input data as a source of *experience* that is available for the program to *learn* to identify structures and patterns of some phenomena of interest. Ultimately, a ML program converts that experience into *expertise* or *knowledge* ([2], Ch. 1, p. 1). This knowledge generally persists in the form of some mathematical structure and remains available for future automation and/or optimisation of specific tasks. This principle is followed in the wide-range applications of current learning programs: from the programs that learn to identify spam emails ([3], [4]) to those that seek to unveil interactions among chemical compounds for drug studies ([5], [6]).

This investigation, however, will not assume any particular application. That is, the methods to be proposed are application-agnostic and they will be concerned with accelerating the execution of machine learning programs under mild assumptions. This means that the resulting learning protocols, in principle, will be general enough to be applied to any application of interest.

1.2 Supervised and Unsupervised Learning

Machine learning programs can be categorised following different principles. They could be separated, for example, according to how computations are performed; that is, we can have *batch learning programs* ([7]) that are assumed to have sufficient resources to store and process their entire input data; and we can have *streaming learning algorithms* ([8]), that usually rely on modest resources to handle large input data, and thus they have to sequentially inspect data points one by one, do some computations, store minimal information, and move on to the next point in the data steam.

A more fundamental distinction among learning programs is that of *supervised learning programs* and *unsupervised learning programs* ([9], Ch. 2 and Ch. 14); the former require that the user provide them with input data that were categorised *a priori*. This categorisation takes the form of *labels* given to each of the input points. This labelling is assumed to be abstractly performed by *the environment*, or by *nature itself*, and hence it is treated as *ground truth*.

Supervised learning programs analyse relations between the data and their labels, and they are expected to output a *rule*, or *hypothesis*, that can be used to assign, with low error, correct labelling for data points that, unlike the input data, have not been labelled before ([10], Ch. 2, p. 21). Thus, in the supervised setting, there are at least two sets of data at play: the input data containing data points along with the labelling information, which is usually referred as the *training set*, and the set of data points that are reserved for assessing the performance of the ML program, the *test set*. We can see that in the supervised setting, the principle of "learning from experience" is followed straightforwardly: the training set contains accumulated experience, and the learning program uses this experience in order to provide the members of the test set the information they are missing: the labelling.

Unsupervised learning programs work differently: they do not distinguish between training and test sets, and their goal is not to *complete* missing information in the data as in the supervised case. Rather, unsupervised learning programs attempt to make sense of the input data by looking for structures among input points, or by obtaining some form of representation of the data ([11]). For example, grouping data points into *clusters* by measuring their closeness is a typical task in the unsupervised setting.

This thesis will be mostly concerned with the supervised setting; in particular, it will cover two fundamental learning methods that belong to such category: *Logistic Regression* and *Support Vector Machines*. Both of these methods will be studied as part of a high-level machine learning framework called *Reliable Machine Learning* (RML). In fact, an important portion of this research work will be devoted to systematically define new learning programs, which will be referred as *learning protocols*, that substantially improve the running time of two important RML techniques: Conformal Prediction and Venn-Abers Prediction.

Unsupervised learning will also play an important role, albeit indirectly, in the contributions of this research. More concretely, the classic k-Means method will be used as a mean to achieve good data representations for logistic regression.

1.3 Reliable Machine Learning

Most of the computational problems considered in this work have their roots in the sub-field of *Reliable Machine Learning* (RML) ([12]): a family of methodologies that supplement the output of machine learning algorithms with sets of reliability measures. This *extra* reliability information can take different forms, depending on the specific methodology at hand. This thesis will cover two fundamental members of the RML family: *Conformal Prediction* (CP) and *Venn-Abers Prediction* (VAP).

CP ([13], Ch. 2, p. 17) allows for the design of supervised-learning programs that not only give the user label predictions for data points he/she is interested in classifying, but also confidence levels for those predictions. Furthermore, the obtained confidence levels constitute strong *prediction error bounds*, which means that the average number of mistakes CP makes can be known *a priori*. Thus, CP is parameterised by the desired *significance level* for its predictions, letting the practitioner carefully control the average number of errors that the CP is allowed to commit.

VAP ([14]) is a relatively new member to the RML family, and it also assumes a supervised-learning setting; however, unlike CP, VAP converts score-based predictions into probability-type results, allowing a wide breadth of non-probabilistic supervised methods to output well-calibrated probabilities. As CP, VAP also provides bounded output; in this case, the output is a tuple containing upper and lower bounds for the *true probability* of observing a specific label for a specific object.

1.4 Problem, Research Questions and Contributions

Now that the computational scenario has been laid out, we are in a good position for presenting our main computational problem. This thesis is concerned with the problem of *scaling up* the methods from RML mentioned above, CP and VAP, to input data of moderate size, as it is well known that the informativeness of these two methodologies comes at the price of high computational overhead (see [13],

Ch. 4, [12], Ch. 2, [15] and [16]). Considerable speed-ups can be obtained for the above methods by assuming an *inductive setting*, and indeed this approach has seen successful applications in domains such as speech recognition ([17]), drug discovery ([5], [18]) and applicability domain determination ([19]). Still, even in the inductive setting, these methods add a non-negligible computational overhead to the learning process, and the data sets used are still restricted to small sizes *i.e.* in the order of few thousands ([18], [20], [21], [22], [15]). Hence, the question remains: *can the running times of CP and VAP be improved in the inductive setting?*

To address the above question, this work proposes a route different from the usual path taken in Computer Science; that is, it will not attempt to improve the running time of CP and VAP by re-designing, or even modifying, these methods. Instead, the main idea will be to systematically replace the input data with a small summarised version of it; and since this summary will be substantially smaller than the original input data, the computations of CP and VAP will inevitably finish faster. In order to obtain a *good* representation of the input data, the algorithmic framework of *coresets* [23] will be used. A coreset (or core-set) is a *compact* data set that *provably correctly* approximates a bigger set of points. Thus, this approach should not be considered as an alternative to the well-known inductive setting; instead, it brings to the table a pre-processing step that can help obtain substantial acceleration in the inductive scenario. Therefore, this thesis will provide detailed experimental analysis on the interaction between methods of RML and coresets, which, to the best of our knowledge, has not been done before.

Naturally, the above idea raises many research questions; a summary of the most fundamental ones can be found below.

***Research Question* 1** (On the validity, efficiency and calibration of CP and VAP)**.** *Assume that it is possible to replace a given data-set with a carefully constructed coreset. Assume further that the obtained coreset is used as input data for (i) a conformal predictor, and (ii) a Venn-Abers prediction instance. Does this compression-based approach meaningfully affect the validity and efficiency for (i) and/or the probability calibration for (ii)?*

***Research Question* 2** (On the computational gains of coreset-based methods for CP and VAP)**.** *Assume that it is possible to replace a given data-set with a carefully constructed coreset. Assume further that the obtained coreset is used as input data for (i) a conformal predictor, and (ii) a Venn-Abers prediction instance. Considering that coresets require extra computations to summarise the given data-set, can coreset-based approaches offer meaningful computational acceleration for scenarios (i) and/or (ii)?*

***Research Question* 3** (On the speed of the coreset computation)**.** *Assume that it is possible to replace a given data-set with a carefully constructed coreset. Does the coreset computation allow for practical optimisations without degrading the coreset quality?*

The above questions converged in a set of methods that provide answers to the stated scientific inquiries. The contributions of this thesis can be summarised as follows:

- **Coreset-based Inductive Conformal Predictors (C-ICP) -** a fast protocol for using conformal predictors over large input data. C-ICP compresses the input data into a small coreset, and runs on it an inductive conformal predictor. The results obtained indicate that C-ICP preserves the validity property of conformal predictors while also retaining the efficiency of the predictions. Therefore, C-ICP provides answers to Research Questions 1 and 2.

- **Inductive Venn-Abers Predictors with Enclosing Balls (IVAP-WEB) -** a coreset-based *calibrator* that improves the running time of Venn-Abers Prediction in the inductive setting. IVAP-WEB uses only a minimal portion of the input data (*e.g.* 1%) to calibrate the predictions of scoring classifiers. The resulting probability estimates are competitive with those obtained using standard Venn-Abers on the full input data, and they can be obtained using only a fraction of the computing time needed when considering all the available

data *e.g.* IVAP-WEB can converge in 10% the time needed by traditional IVAP. Therefore, IVAP-WEB allows us to answer Research Questions 1 and 2.

- **Accelerated Clustering via Sampling (ACvS) -** an acceleration strategy for computing clustering-based coresets. Based on the proven fact that, in expectation, a uniform random sample provides unbiased estimation for some instances of clustering, ACvS proposes to construct a coreset by clustering only a small uniform random sample taken from the input data. We show that when using ACvS, the resulting coreset performs exactly the same as the coreset obtained from clustering over the full input data, but the coreset construction is meaningfully accelerated. ACvS sheds light on Research Question 3.

- **Regressed Data Summarisation Framework (RDSF) -** a coreset-based data compression framework that not only reduces the size of a data-set, but also *learns* to distinguish important points, in a well-defined sense, from irrelevant ones. This knowledge is preserved in the form of a regression model, and can be used to identify the importance of new points that may become available to the practitioner, without the need of running the full coreset algorithm. RDSF also contributes an answer to Research Question 3, and leaves the door open for extensions to other coreset construction algorithms, and to different computational settings *e.g.* streaming.

1.5 Thesis Structure

The thesis follows the following structure:

- **Chapter 2** presents the machine learning methods and definitions necessary for understanding this work's new learning protocols. A discussion on suitable data for said protocols will also be presented in this chapter, and such discussion will culminate with the group of data-sets chosen for demonstrating their effectiveness.

- **Chapter 3** provides a in-depth introduction to the central computational technique over which the contributions of this thesis are built: the paradigm of coresets.

- **Chapter 4** introduces the theory of Conformal Prediction, and presents the first contribution of this work: Coreset-based Inductive Conformal Prediction (C-ICP).

- **Chapter 5** defines the framework of Venn-Abers Prediction and presents the second contribution of this work: Inductive Venn-Abers Predictors with Enclosing Balls (IVAP-WEB).

- **Chapter 6** identifies a computational bottleneck in the state-of-the-art coreset algorithm for the Logistic Regression learning method, and in response proposes two accelerating techniques to circumvent the problem: the Accelerated Clustering via Sampling (ACvS) procedure and the Regressed Data Summarisation Framework (RDSF), which are the third and fourth contributions of this research, respectively.

- **Chapter 7** concludes the thesis, and discusses future problems.

Where is the wisdom we lost in knowledge? Where is the knowledge we lost in information?

— T.S. Eliot
The Rock. ([24])

2 Machine Learning Methods

2.1 Introduction

The first formal topic that needs to be addressed in order to present the contributions of this thesis emerges from the following question: "what are the patterns that we are interested in observing in the input data?". As stated in the previous chapter, machine learning programs are particular computer programs that attempt to discover patterns in their input data in automated fashion. The patterns that they may find is generally restricted to the specific *machine learning algorithm* they implement. Thus, depending on the machine learning algorithm used, different patterns emerge from the input data.

This chapter presents a set of machine learning algorithms that assume different settings and work under different assumptions, to ultimately find structures in the data, and *learn* those structures. Primarily, the most important algorithms for presenting our work are two fundamental methods to perform object classification: *Logistic Regression* (LR) ([25]) and *Support Vector Machines* (SVM) ([26]). The main objective of this thesis is to accelerate the computation of the algorithms used to solve these two problems for cases where the input data become large; specially, we are interested in scenarios where LR and SVM are cast as part of the machine learning frameworks of Conformal Prediction (CP) and Venn-Abers Prediction

(VAP). These two machine learning frameworks will be presented in Chapter 4 and Chapter 5, respectively. However, the definition of LR and SVM do not depend on these frameworks. Furthermore, once the two machine learning problems have been formally defined, their connection with CP and VAP will become clear.

Our contributions rely on obtaining computational acceleration by summarising, or compressing, the input data for the problems of LR and SVM. The specific data compression strategy we will employ will be discussed in Chapter 3; however, for the compression process, we will need additional tools from machine learning. Specifically, we will use the methods of *K-Means clustering* ([27]) and *Ordinary Least Squares* ([2], Ch. 9, p. 95) for obtaining a grouping of the input data and a regressor, respectively. Both methods will be formally addressed and defined in this chapter, and their connection with data compression will become clear in the next ones.

Finally, the chapter will conclude with the specifications and descriptions of the data-sets that will be used to evaluate our machine learning programs in Chapters 4, 5 and 6.

2.2 The Big Data Paradox

We start the exposition of the machine learning methods that this thesis is concerned about with an important phenomenon that takes place when dealing with machine learning programs. There are two competing forces at play whenever we are attempting to *learn* over some input data.

On one hand, ML algorithms benefit substantially from having large input data. In fact, the theoretical success of most machine learning methods generally depends on having a *sufficiently large* input data, to ensure bounded generalisation error. In statistical learning theory, this *sufficiently large* condition is referred as *the sample complexity*, and it states that the larger the input data, the more probable that the learning methods will produce correct results. A general bound on the sample complexity can be found in [2], Chapter 3, Corollary 3.2.

On the other hand, having a large amount of input data does create computational bottlenecks in the learning process as most ML programs can quickly exhaust

the available computing resources. Therefore, learning algorithms improve as the amount of available data increases, and, at the same time, their computability and convergence dramatically deteriorate with the increase of available information. We refer to this set of conflicting objectives as the *big data paradox*, and we will see this phenomenon manifesting itself very often in our discussions. Our work, as will be presented in Chapters 4, 5 and 6, reacts to this issue by making a compromise on the accuracy of machine learning algorithms, in a well-defined sense, for the sake of computational efficiency.

In the next section we define the general setting for classification problems in ML, and instantiate two fundamental methods: Logistic Regression and Support Vector Machines.

2.3 Classification

An elemental task in machine learning is to make predictions. Any ML program that makes predictions poses, either implicitly or explicitly, the following question: having observed a set of objects labelled in certain fashion, can we extract, or construct, *a rule*, or *model*, to predict the labelling for objects that (i) have not been observed before, (ii) do not possess any label assigned to them?

We can formalise the above scenario as follows: let the input data be of the form $\mathcal{D} := \{(x_i, y_i)\}^n$, where n is the total number of (x_i, y_i) tuples in $\mathcal{D}$, x_i is an *object*, or a *feature vector*, that mathematically belongs to the *object space* $X := \mathbb{R}^d$, and y_i is the *label* for x_i, and it belongs to the *label space* Y. Sometimes, it is useful to refer to a tuple $(x, y) \in \mathcal{D}$ as an *input example*, where each input example $z := (x, y)$ comes from the space formed by the Cartesian product $Z := X \times Y$.

Depending on the mathematical form of the label space, prediction problems in machine learning can be instances of a *classification problem*, where Y is a finite set and each member of Y is called *a class*; or a *regression problem*, where Y is the infinite set of the real numbers *i.e.* $Y := \mathbb{R}$. In this thesis we will be concerned with classification problems that assume $|Y| = 2$; specifically, we mathematically define the label space as the set $Y := \{-1, 1\}$. Classification problems of this kind

are called *binary classification problems*, and they are of fundamental importance in machine learning as every classification problem can be reduced to a set of binary classification problems [1]. It is useful to mention at this point that for binary classification problems, the label $1 \in Y$ is referred as *the positive label*, whereas the remaining label, $-1 \in Y$ in our definitions, is called the *negative label*.

A fundamental assumption for our machine learning setting is the *Randomness Assumption*, which we state as follows:

Assumption (Randomness)**.** *Every input example $z \in \mathcal{D}$ is sampled independently from some probability distribution on the space Z.*

Such assumption is a standard one in learning theory, and it is equivalent to stating that all examples in $\mathcal{D}$ are **i**ndependently and **i**dentically **d**istributed (IID).

A rule to make predictions in the above scenario is called a *hypothesis*, a *classifier* or a *predictor* [2], and it can be mathematically defined as a function $h_\theta : X \to Y$, where h is parameterised by θ, which is called *the model* or the *learning parameter*. That is, the predictor h maps an object to its potentially correct label, hence *predicting* what the label of an object x is, considering the patterns seen in the input data. We refer to the predicted label for a certain object x as $\hat{y} := h_\theta(x)$, to distinguish it from the ground-truth label y.

The above definition of h is an example of a *parametric classifier*: a classifier that looks into the available input data and keeps a model, the (learning) parameter θ, that mathematically reflects the patterns seen in the data. There are different methods for obtaining the model θ, and each method defines a different classifier. We shall consider two important methods for obtaining a binary classifier: LR and SVM. Each method allows for the design of a hypothesis h by providing different procedures for obtaining the underlying model θ which, mathematically, for both of these problems, is of the form $\theta \in \Theta$, where $\Theta := \mathbb{R}^{d+1}$.

[1] There are standard procedures to do this. Two important approaches are: "one-against-all" and "one-against-one". See [28] for details.

[2] the more appropriate terms are *scoring classifier* and/or *single-point predictor*. The rationale for this naming convention will become clear once Conformal Prediction and Venn-Abers Predictors are introduced in Chapters 4 and 5, respectively.

2.3.1 Logistic Regression

The Logistic Regression method proposes to obtain a learning parameter for the classifier h by optimising an objective function called the *log-likelihood function*, which we shall define in this section. For an unlabelled object x, the LR classifier makes the prediction $h_\theta^{LR}(x) = \sigma_{logistic}(x \cdot \theta)$, where $\sigma_{logistic}$ is defined as follows ([2], Ch. 9, p. 97):

$$\sigma_{logistic}(r) := \begin{cases} 1 & 1/(1+\exp(-r)) > 0.5 \\ -1 & \text{otherwise,} \end{cases} \tag{2.1}$$

where $r \in \mathbb{R}$ and the expression $1/(1+\exp(-r))$ is a *sigmoid function* [3] known as the *logistic function*. As we mentioned earlier, the challenge is to find the parameter $\theta \in \Theta$ that the LR classifier relies upon for making predictions.

We assume we have the input data $\mathcal{D} := \{(x_i, y_i)\}^n$; we further define the *likelihood* of observing $y_i = 1$ for x_i as $p_{logistic}(y_i = 1|x_i; \theta) := 1/(1+\exp(-x_i' \cdot \theta))$, where $\theta \in \Theta$ is the learning parameter and $x_i' \in \mathbb{R}^{d+1}$ is the object x_i extended with an *1* element to account for *the bias term*. The latter is a natural procedure for writing an affine function in $\mathbb{R}^d$ as its homogeneous linear counterpart in $\mathbb{R}^{d+1}$ ([2], Ch. 9, p. 90).

Similarly, we can write the likelihood of observing the label $y_i = -1$ for object x_i as:

$$\begin{aligned} p_{logistic}(y_i = -1|x_i; \theta) &:= 1 - \frac{1}{(1+\exp(-x_i' \cdot \theta))} \\ &= \frac{\exp(-x_i' \cdot \theta)}{(1+\exp(-x_i' \cdot \theta))} \\ &= \frac{1}{(1+\exp(x_i' \cdot \theta))}. \end{aligned} \tag{2.2}$$

Therefore, for any y_i, we can write its likelihood as $p_{logistic}(y_i|x_i; \theta) := 1/(1+\exp(-y_i x_i' \cdot \theta))$.

Because the input data $\mathcal{D}$ is assumed to be *independent and identically distributed*, an assumption we will carry throughout the whole thesis, the log-likelihood function $LL_n(\theta|\mathcal{D})$ can be defined as ([2], Ch. 9, p. 98):

[3] "Sigmoid" means "S-Shaped", and it refers to the shape the function takes for all values in its domain.

$$LL_n(\theta|\mathcal{D}) \;:=\; \sum_{i=1}^{n} \ln\; p_{logistic}(y_i|x_i;\theta) \;=\; -\sum_{i=1}^{n} \ln(1 \;+\; \exp(-y_i x_i' \;\cdot\; \theta)), \quad (2.3)$$

which is the objective function for the LR problem. The optimal solution for the above function can be obtained using *maximum likelihood estimation* ([29], Ch. 4, p. 200). Maximising $LL_n(\theta|\mathcal{D})$ is equalivent, however, to minimising $\mathcal{L}_n(\theta|\mathcal{D}) := \sum_{i=1}^{n} \ln(1 + \exp(-y_i x_i' \cdot \theta))$ over all $\theta \in \Theta$. With this manipulation in place, the optimisation problem for finding an optimal LR model can be written as:

$$\theta^* := \arg\min_{\theta \in \Theta} \mathcal{L}_n(\theta|\mathcal{D}). \quad (2.4)$$

Once we have optimised Equation (2.4), we can use the optimal θ^*, or an *approximation* to it, to define the LR classifier $h^{LR}_{\theta^*}$ to predict the labels of *unseen* data *i.e.* unlabelled objects that were not used in Equation 2.4 to find the optimal model for LR. Thus, for an unlabelled object x, the LR classifier we just defined will make a prediction $h^{LR}_{\theta^*}(x) = \sigma_{logistic}(x' \cdot \theta^*) = \hat{y}$, where $\hat{y} \in Y$.

The advantage of the LR hypothesis is that, because of the nature of the logistic function, it provides a probabilistic interpretation to its predictions; that is, the values produced by the logistic function, specified in the definition of $\sigma_{logistic}$ in Equation (2.1), can be interpreted as probabilities for seeing the positive label. Furthermore, these probabilities are generally *well-calibrated* in practice ([30]). Therefore, the LR classifier concludes that, if the logistic value is greater than 0.5, then it is more likely that the label for the object x is 1; if, however, the logistic value is less or equal to 0.5, then it is safer to predict the negative label.

The log-likelihood function in Equation (2.3) is a *convex function*, and thus modern solvers can efficiently optimise it ([2], Ch. 12, p. 124). However, as the number of examples in the input data $\mathcal{D}$ grows, finding the optimal model θ^* becomes computationally prohibitive, specially if the LR model is computed within a machine learning framework such as Conformal Prediction. This topic was addressed extensively in our work published in [31]. By inspecting Equation (2.3) closely, we indeed see that the number of computations done grows at least linearly

with the number of input points. As we will see in Chapters 4 and 6, our main claim is that it is not necessary to compute the log-likelihood for every input point in order to obtain a good LR classifier; it is enough to perform the computations on a very small subset of the data. By reducing the number of operations in this way, we can efficiently obtain an *approximated* LR model, $\bar{\theta}$, and show that the resulting LR classifier $h_{\bar{\theta}}^{LR}$ performs well both within and without the Conformal Prediction framework. In Chapter 3 we will present the computational technique for finding such sub-sets of input points.

2.3.2 Support Vector Machines

We now turn our focus toward another approach for defining a classifier h_θ: the *Support Vector Machines* (SVM) paradigm ([26]).

The SVM method attempts to find a linear separation in the input data by finding a *hyperplane* [4] that does not only separate input points, but also keeps them far from itself ([2], Ch. 9, p. 90). This idea is formalised with the concept of a *margin*, which is the minimal distance between a point in the input data and a hyperplane ([2], Ch. 15, p. 168). Thus, SVM finds hyperplanes that separate the input data by a *large margin*. A classifier defined using the SVM paradigm takes the form of $h_\theta^{SVM}(x) = sign(x' \cdot \theta)$, where $x' \in \mathbb{R}^{d+1}$ includes the bias term and the function $sign(a)$ outputs 1 if $a > 0$ and -1 otherwise. Thus, $h_\theta^{SVM}(x)$ uses a large-margin hyperplane θ to make its predictions.

The SVM paradigm allows to be instantiated under different assumptions, depending on what knowledge we might have regarding the input data. For example, we can assume that the input data is *linearly separable* and, under that assumption, we would have an instance of a *hard-SVM* formulation ([9], Ch. 12, p. 417-420). However, linear separation is a strong assumption which can prove restrictive for the applicability of our contributions. Thus, we consider the *soft-SVM* formulation which allows for non-separable input data while attempting to find

[4] any hyperplane can be written as the linear function $l(x) := \sum_i x_i\theta_i = x \cdot \theta$, where x and θ are vectors, and the x_i and θ_i are their respective components. In our ML context, we consider x to be an input point and θ the learning model.

hyperplanes that have as few input points as possible laying on the wrong side of the halfspace. Formally, for our input data $\mathcal{D}$, obtaining a model θ with the soft-SVM formulation is equivalent to minimising the following objective function over all $\theta \in \Theta$ ([2], Ch. 15, p. 172):

$$f(\theta) := \lambda||\theta||^2 + \frac{1}{n}\sum_{i=1}^{n} \ell(\theta; x_i, y_i), \tag{2.5}$$

where $\ell(\theta; x_i, y_i) := \mathbf{max}\{0, 1 - y_i(x_i' \cdot \theta)\}$ is a *loss function* ([32]) called the *hinge loss* and $\lambda > 0$ is the parameter determining, or regularising, the trade-off between increasing the margin size and ensuring that training data lie on the correct side of the margin.

By definition, minimising the function in Equation 2.5 will allow us to find a hyperplane θ^* that makes a large-margin *linear* separation among input points. The SVM paradigm, however, can naturally be extended to also capture non-linearity among input points, which in turn could enhance the generalisation capabilities of the hypothesis $h_{\theta^*}^{SVM}$. This can be achieved via the use of *kernels* ([9], Ch. 12, p. 417): mappings to high-dimensional feature spaces that allow the SVM classifier to cope with non-linearity; that is, data points are mapped from their original *input space* to a *feature space* where the former is embedded, so that a linear separator can be learned in the latter space.

We can formalise this as follows: let $K(w, z) := \exp(-\gamma||w - z||^2)$ and $\phi(\cdot)$ be the *Radial Basis Function* (RBF) kernel ([33]) and the feature mapping induced by the kernel, respectively, with $w, z \in \mathbb{R}^d$ and $\gamma \in \mathbb{R}$. The RBF kernel function, also known as the Gaussian kernel function, allows us to obtain the dot product for any two vectors without computing explicitly their mapping to a higher dimensional space. Therefore, we can equivalently represent the kernel through the corresponding kernel function and via its induced feature mapping *i.e.* $K(w, z) := \phi(w) \cdot \phi(z) := \exp(-\gamma||w - z||^2)$.

We can then re-write Equation (2.5) as

$$f(\theta) := \lambda||\theta||^2 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{max}\{0, 1 - y_i(\theta \cdot \phi(x_i'))\}, \tag{2.6}$$

which now applies the mapping ϕ to all x_i.

Equation 2.6 shows the *kernelised* SVM objective function in the *primal form*. We shall use this formulation for the rest of the thesis. Finally, the optimisation problem for the just defined SVM formulation is:

$$\theta^* := \arg\min_{\theta \in \Theta} f(\theta), \tag{2.7}$$

where θ^* is guaranteed to be a large-margin separator.

An important property of the solution for Equation (2.7) is that, following a result known as the *Fritz John optimality conditions* ([34]), the SVM solution can be written as a linear combination of the input points ([35]); that is, the optimal SVM solution is of the form $\theta^* := \sum_{i=1}^{n} \alpha_i y_i \phi(x_i)$ ([33]). The α_i are real-valued coefficients, and when they have a non-zero value, their companion vectors x_i are called *support vectors* [5]. Based on this property, it is standard to write the primal SVM objective function, defined in Equation (2.6), in its *dual form* [6]. The dual SVM formulation directly optimises over the α coefficients that can be used to construct the optimal solution. For the sake of completeness, we include the dual SVM formulation as defined in [2], Ch. 15, p. 176:

$$\max_{\boldsymbol{\alpha} \in \mathbb{R}^n : \boldsymbol{\alpha} \geq 0} \left(\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \right), \tag{2.8}$$

where $\boldsymbol{\alpha} \in \mathbb{R}^n$ is a column vector containing the α coefficients for each input point, and $K(\cdot, \cdot)$ is the kernel function that computes the dot product for any two input points. Having both the primal and the dual formulation for SVM, it is important to state that, for our work, the primal SVM objective function is preferable as (i) it allows for gradient-based optimisation solvers such as the *Stochastic Gradient Descent* (SGD) ([37]); (ii) it can be naturally extended to online or streaming settings; (iii) it does not suffer from the over-quadratic computational complexity of the dual form ([38]).

[5] since the SVM solution θ^* is in the *linear span* of the vectors x_i with coefficients $\alpha_i \neq 0$, the x_i vectors are said to geometrically *support* the optimal solution.

[6] the dual form can be derived from the primal by applying a tool from optimisation known as the Karush-Kuhn-Tucker theorem ([36]).

Still, solving the primal SVM poses a hard computational challenge as the number of input points increases. This complexity arises from the fact that, when using kernels, the number of non-zero α_i, and thus the number of support vectors, grow almost linearly with the number of input points ([39]). This phenomenon is commonly known as the *curse of kernelization* ([40]), and the computational challenges it presents are well-known and well documented (see [41], [40], [35] and references therein).

To mitigate the curse of kernselisation, sparse models are generally preferred as solutions for Equation (2.7) as sparsity, in this context, means that less input points contribute toward the model; and hence the optimisation problem can be solved faster. However, in alignment with the big data paradox discussed previously in this chapter, too much sparsity can damage the quality of the obtained model. In Chapter 3 we will present a tool that, heavily relying on results and intuitions from the field of computational geometry, will allow us to obtain sparse SVM models with strong convergence guarantees.

2.4 Other Machine Learning Problems

This section presents additional tools from machine learning whose definitions are important for understanding our contributions in Chapters 4, 5 and 6.

2.4.1 k-Means Clustering

k-Means is an instance of clustering, an important unsupervised learning scenario that was briefly mentioned in Chapter 1. Webster ([42]) intuitively defines clustering as "a statistical classification technique for discovering whether the individuals of a population fall into different groups by making quantitative comparisons of multiple characteristics". Thus, the aim of any clustering algorithm is to find the natural grouping of input points based on one or more well-defined similarity measures.

Generally, ML programs that implement a clustering algorithm are parameterised by an integer k, which is the number of groups, also known as *clusters*, that are to be found in the input data. We are interested in using the k-Means method

([27]) for obtaining a clustering of the input data. A solution found with k-Means produces a partition of the input data, where each of the parts is concentrated near their central point: the *centre*. Thus, the problem consists in finding those centres, which ultimately will induce the desired clustering over the input data. Formally, let our input data be of the form $\mathcal{D} := \{(x_i)\}_{i=1}^{n}$ and let $k > 0$ be the number of clusters that we want to find in the input data. Notice that since clustering assumes an unsupervised ML setting, our input data is unlabelled. Let $Q := \{q_1, q_2, ..., q_k\}$ be a set of k potential cluster centres *i.e.* $|Q| := k$, and define $\mathcal{Q}$ as the set of all sets containing k points in $\mathbb{R}^d$. The k-Means method attempts to optimise the following objective function ([43]):

$$Q^* = \arg\min_{Q \in \mathcal{Q}} cost(\mathcal{D}, Q) := \sum_{x \in \mathcal{D}} \min_{q \in Q} ||x - q||_2^2 \tag{2.9}$$

In this context, Q is *a query* and $\mathcal{Q}$ is the *query space*. Thus, each query has a computable *cost* associated with it; in the case of k-Means, the sum of *squared Euclidean distances* to the input points is the cost, and the task is to find a set of k centres, Q^*, that minimises that cost. Therefore, the similarity measure used to obtain a k-Means clustering of the data is the squared Euclidean distance. If we change this measure to the Euclidean distance *i.e.* we replace $||x-q||_2^2$ with $||x-q||_2$ in Equation (2.9), we would have an instance of the k-Median method ([44]).

The state-of-the-art procedure that we will rely upon for solving the above optimisation problem is called k-Means++ ([27]). In order to describe the steps performed by this procedure for solving Equation (2.9), let $D(x, S)$ be the minimum squared Euclidean distance between point x and any set $S \subseteq \mathbb{R}^d$ *i.e.* $D(x, S) := \min_{s \in S} ||x - s||_2^2$.

1. initialise an empty set $Q = \emptyset$ to collect the centres.

2. choose the first centre q_1 uniformly at random from the input data $\mathcal{D}$. Add it to Q *i.e.* $Q = Q \cup \{q_1\}$

3. take a new centre q_i from the input data, choosing $x \in \mathcal{D}$ with probability $\frac{D(x,Q)}{\sum_{x \in \mathcal{D}} D(x,Q)}$. Add q_i to Q *i.e.* $Q = Q \cup \{q_i\}$.

4. repeat step 3 until k centres are sampled *i.e.* until $|Q| = k$ holds.

5. for each $i \in \{1, 2, ..., k\}$, set the cluster C_i to be the set of points in $\mathcal{D}$ that are closer to centre $q_i \in Q$ than they are to centres $q_j \in Q$, with $i \neq j$.

6. for each $i \in \{1, 2, ..., k\}$, set $q_i \in Q$ to be the mean of all points in C_i: $q_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$.

7. repeat steps 5 and 6 until Q no longer changes.

David Arthur and Sergei Vassilvitskii proved in [27] that the above sequence of steps find a query $\bar{Q}$ that is $O(log(k))$ competitive with the optimal solution Q^* *i.e.* $cost(\mathcal{D}, \bar{Q}) \leq O(log(k))cost(\mathcal{D}, Q^*)$. This idea of initialising a k-means solution with a set of centres that is picked non-uniformly at random (steps 3 and 4) will be fundamental to understand the compression techniques to be discussed in Chapter 3.

2.4.2 Linear Regression

We have mentioned before that an instance of a prediction problem where the label space Y is the set of real numbers is called a regression problem. As we will see in Chapter 6, regression will be used by one of our protocols to provide valuable information during and after a data compression process.

In particular, we will consider the case of linear regression. Let our input data be $\mathcal{D} := \{(x_i, y_i)\}^n$, where the object $x_i \in X$ has a *target value* $y_i \in Y$, with $Y := \mathbb{R}$. An algorithm that makes predictions in a regression setting is called a *regressor*.

The *Ordinary Least Squares* (OLS) method finds a regressor to the following objective function ([2], Ch. 9, p. 96):

$$\arg\min_{\theta \in \mathbb{R}^{d+1}} \frac{1}{n} \sum_{i=1}^{n} ((\theta \cdot x_i') - y_i)^2 \tag{2.10}$$

where as before, $x_i' \in \mathbb{R}^{d+1}$ is the object $x_i \in \mathbb{R}^d$ padded with a 1 to account for the bias term. Therefore, the OLS approach consist in finding a a regression model θ^* that minimises the discrepancy between its prediction for object x_i and the ground-truth target value y_i, for all i. The most attractive aspect of this

formulation is that it is not necessary to use an optimisation solver to obtain a solution because Equation 2.11 allows for a *closed-form* solution. Let $\boldsymbol{X}$ be a matrix containing all the feature vectors $x' \in \mathcal{D}$ as rows, and $\boldsymbol{y}$ by a column vector containing all the true target values for the vectors in $\boldsymbol{X}$. Then the OLS solution can be exactly computed as ([9], Ch. 1, p. 12):

$$\theta^* = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{2.11}$$

As anticipated, we will efficiently make use of this θ^* in Chapter 6 in the context of compressing input data.

2.5 Input Data

This section ends the chapter with a discussion on the *kind* of input data that are suitable for the learning protocols that will be proposed in the thesis. As Section 1.1 mentions, a key advantage of the new protocols is their generality. It is fundamental, however, to explicitly state what "generality" means when referring to input data. There are two assumptions on the input data that the new protocols require in order to work. These assumptions can be considered as mild ones as they are standard for machine learning methods.

The first assumption that needs to hold for the input data of the new protocols is that the number of dimensions of the input points, d, should be *strictly smaller* than the number of available input points, $|\mathcal{D}| := n$ *i.e.* $d < n$ must be true. Hence, the input data should ideally be represented by a *tall and skinny matrix* ([45]). The rationale for this assumption is that the data compression techniques the new learning protocols will deploy are strongly investigated under this assumption. Furthermore, the data used in Conformal Prediction conform well with this kind of data. It follows, then, that the new protocols will be suitable for applications where the norm is to have a large number of relatively low-dimensional records, such as chemical compounds interactions detection ([46], [47]), drugs discovery ([5], [18]) and indoor positioning via signal processing ([48]), to name a few. It is worth

mentioning that an important *group* of applications where this assumption is mostly violated is that of text processing/mining (see [49] and references therein), where the data is traditionally represented with *short and fat matrices* ([50]).

The second assumption is that data points in the input data should be *independent and identically distributed* (IID) with respect to some probability distribution. As will be discussed in Chapter 4, this assumption is crucial for the Conformal Prediction method. Furthermore, most machine learning applications, including all the ones cited previously, need this assumption in order to *learn* anything. A new machine learning set of applications where the IID assumption *does not* hold is that of *Federated Learning* ([51]). Thus, the data used in such setting cannot be used with the protocols proposed in this thesis.

2.5.1 Data-sets

To finish the above discussion on the nature of the input data suitable for the protocols proposed in this thesis, Table 2.1 summarises the main characteristics of the data-sets selected for testing the new methods. This table will be particularly useful later as the contribution chapters will refer to it every time a set of experiments results is presented.

Table 2.1: The six data-sets that will be used to evaluate the new methods proposed in this thesis. **N** is the number of examples, **d** is the number of dimensions for each example; + shows the number of positive examples, **-** is the number of negative examples and +/- shows the rate of the former with respect to the latter. **Spa** stands for the sparsity of the data-sets.

Data-set	**N**	**d**	**+**	**-**	**+/-**	**Spa**
covertype ([52])	581,012	54	283,301	297,711	0.9	0.77
webspam ([3])	350,000	254	137,811	212,189	0.6	0.66
higgs ([53])	11,000,000	28	5,170,877	5,829,123	0.8	0.07
a9a ([54])	48,842	123	37,155	11,687	3.1	0.88
ijcnn1 ([55])	141,691	22	128,126	13,565	9.4	0.40
w8a ([54])	64,700	300	62,767	1,933	32.4	0.96

The selected data-sets are widely used for testing data compression techniques and for ML classification in general (see their sources and their corresponding

citations). Furthermore, each data-set is publicly available at their respective sources. The data-sets can also be downloaded from the well-known *LIBSVM* [7] and *UCI Machine Learning* [8] repositories.

As mentioned earlier in this chapter, this work focuses specifically on binary classification. Then, for each data-set in Table 2.1, the following information is shown:

- Volume of Data (**N** column): the total number of examples available from the data-set. To apply ML methods, a percentage of this number is used for the learning process while the rest is used for testing learning performance. Concrete percentages will be provided in the experiments sections in the contribution chapters.

- Dimensionality of Data (**d** column): the number of variables considered in the data-set. Mathematically, this is the number of dimensions for each feature vector in the data.

- Number of positive examples (+ column): total number of examples (x, y) for which $y = 1$.

- Number of negative examples (**-** column): total number of examples (x, y) for which $y = -1$.

- Balance rate (+/**-** column): a score that assesses how *balanced* ([56]) the data-sets are in terms of the number of positive and negative examples. Concretely, it shows the proportion of the number of positive examples with respect to the number of negative examples, for each data-set. Thus, a value greater than 1 means an *imbalance* in favour of the positive class, while a value smaller than 1 means that there are more examples belonging to the negative class than those that belong to the positive class. Therefore, the closer this score is to 1, the more balanced the data-sets are. In similar measure, the further it is from 1, the more imbalance we have in the data.

[7] https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/ - last accessed 15/11/2021.
[8] https://archive.ics.uci.edu/ml/index.php - last accessed 15/11/2021.

- Sparsity score (**Spa** column): sparsity is an important structure in the data; it tells the proportion of the number of *zero* elements in the feature vectors with respect to the number of *non-zero* entries. This metric measures sparsity by dividing the count of zero values in the feature vectors' elements in a data-set over the total number of elements for all the feature vectors ($\mathbf{N} * \mathbf{d}$). Notice that the conceptual opposite to sparsity is *density*; and thus a density score could be computed by performing 1 - **Spa**.

Regrading the sources for these data-sets, they arise from a wide variety of applications. Then, these applications are also suitable for this thesis' protocols and can be considered alongside those mentioned in the previous sub-section. Each data-set in Table 2.1 can be described as follows:

- covertype: a database built at the College of Natural Resources from the Colorado State University. It contains cartographic variables that are used to predict different forest cover types. Originally, the data-set contains 7 possible forest cover types. However, in our work we use the binary version of the data-set in which the 7 cover types are combined into two possible classes: cover type 1 and cover type 2.

- webspam: a large corpus of emails structured in variables such as average length of uninterrupted sequences of capital letters, number of appearances of specific *spam* words, etc. Each row, or object, in the corpus is appropriately labelled as *spam* or *not spam*. This data-set is also known as spambase, and it has been extensively used in the design of spam filters (see [3]).

- higgs: a physics data-set produced by a mixture of measurements made by in particle accelerators and Monte Carlo simulations. Specifically, the first 21 variables are proper measurements and the last 7 are simulations based on those 21 measurements. The task is to predict whether a Higgs boson will be detected (signal 1) or not (signal 0). More details can be consulted in [53].

- a9a: a database that contains census information for adult people, including variables such as age, work class, education, marital status and more. The task is to predict whether their yearly income will exceed $50,000. This data-set has been used to show the performance of the *Sequential Minimal Optimization* SMO algorithm ([54]) for learning SVM classifiers.

- w8a: data-set designed in the context of text categorisation: the data contains as variables different keywords found in web pages and the task is to predict whether the web pages belong to a category or not. Similar to a9a, this data-set has also been used to test the effectiveness of the SMO algorithm for the problem of support vector machines.

- ijcnn1: synthetic data-set used as part of a machine learning challenge during the first edition of the well-known *International Joint Conference on Neural Networks* (IJCNN 2001) ([55]). ijcnn1 presents a complex scenario for learning classifiers due to the imbalance found in the data *i.e.* about 90% of the instances belongs to the positive class.

2.6 Chapter Summary

Machine learning is the field concerned with automated detection of patterns in data. One of the most fundamental tasks in the field is that of classification. Two of the most important classification methods are logistic regression (LR) and support vector machines (SVM). LR allows for the design of a classifier by finding a model that maximises the likelihood of predicting the correct labelling for a set of training examples. From the optimisation point of view, this task is equivalent to minimising the negative log-likelihood function. The resulting classifier makes predictions that have some interesting probabilistic interpretations, and the LR classifiers uses a threshold of 0.5 to decide if the considered object belongs to the positive or negative class.

SVM proposes a different formulation for finding a classification model. Based on the idea of large-margin separation, SVM minimises the hinge loss in order to

obtain a hyperplane that not only separate input points from different classes, but also separates them by a large margin. The predictive capability of SVM can be enriched with the use of kernels, which are functions that map input points to higher dimensional spaces. It is a standard result in machine learning that by using kernels, non-linearity in the input space can be captured by a linear classifier in the high-dimensional feature space where input points are embedded. This powerful feature, however, does not come for free as high computational cost, in terms of both computing power and storage, is incurred with the use of kernels. This issue will be a central topic to two of our contribution chapters.

We have also presented the method of K-Means clustering, which will be useful for understanding the data compression technique we will present in the next chapter. Similarly, the problem of finding a linear regressor will be fundamental for understanding one of our learning protocols, and thus, a gentle introduction was provided as a reference.

Finally, this chapter showed the data-sets chosen to present the empirical results in this work. Each data-set is publicly available and can be easily obtained for replication purposes. As long as the application area of interest uses data that (i) can be represented with a tall and skinny matrix; (ii) have IID input data points, the learning protocols in this thesis will be effective.

Truth is much too complicated to allow anything but approximations.

— John Von Neumann
The Neumann Compendium. ([57])

3 Coresets

3.1 Introduction

The previous chapter has presented a set of machine learning techniques for finding patterns in data and it has also highlighted the importance of having large amounts of information as input for the machine learning algorithms used in those methods; in other words, it has established that larger input data generally translates into better learning. The computational consequence of this necessity, however, is that the running time of machine learning programs can exhaust the available computing power very quickly. We have referred to this phenomenon as *the big data paradox*.

This chapter introduces the central algorithmic tool this thesis uses for finding middle-ground solutions to the above dilemma: the framework of coresets. Resembling the concept of approximation algorithms in Computer Science, this technique allows for the computation of a *good* solution as a surrogate to an *optimal* solution, for the sake of computational efficiency, for hard computational problems. Unlike conventional approximation algorithms, however, the coreset framework does not attempt to change the algorithms whose running times we wish to improve; instead, it aims to reduce, or *compress*, their input data into a small *summary* of data: the coreset. Therefore, since computational complexity is measured as a function

of the input size, we will aim to obtain computational acceleration by replacing the large input data with a its compressed version.

The most attractive feature of this data-compression technique is that the coreset theoretically preserves the main structures of the large input data, with respect to a well-defined mathematical function; and thus, once a coreset is computed, it can be used as a reliable *proxy* to the input data. From the statistical-learning point of view, much like the training set can be seen as a *window* through which a partial view to the ground-truth data-generating mechanisms becomes available to ML algorithms, the coreset can be understood as a *small* window to the larger input data it represents.

Coresets will be instrumental to our contributions in Chapters 4, 5 and 6; therefore, we devote this chapter to presenting the formal definitions for this data-compression technique, along with the algorithms that will be used for computing a coreset for the ML scenarios considered in the next chapters.

3.2 Background

The idea of approximating the solution of an existing algorithm is not new in Computer Science; in fact, approximation algorithms are the standard approach for tackling *hard* computational problems such as the set cover problem ([58]), scheduling ([59]) and the knapsack problem ([60]).

The necessity of finding approximated solutions as surrogate for the optimal ones is rooted in the inherent computational complexity of many problems. To remind the reader, there are two main *classes* of problems in Computer Science: those that can be solved in computational time that is *polynomial* in the input size are said to belong to class **P**; on the other hand, the problems that (currently) do not allow for a solution in polynomial time are contained in class **NP**. An algorithm converges in polynomial time if and only if said time can be written as a function $f(n) = n^k$ ([61], Ch. 7, p. 286), where n is the input size and k is a constant. Thus, in Computer Science, problems that allow polynomial-time solutions are regarded as *efficiently solvable*, or in computational terms, *tractable*.

The design of algorithms becomes much more challenging for the problems in the **NP** category as no polynomial-time solutions are known for them. In this case, the algorithm designer has two options: (i) attempt to use an existing inefficient algorithm, if any, to design a polynomial-time solution, or (ii) design an algorithm for a *relaxed* version of the problem and hence accept some compromises in terms of quality of the solutions, generality of the solutions or reliability of the solutions. Approach (i) involves solving the most important computational problem of our time, because finding a polynomial-time solution for a problem in **NP** immediately implies that $\mathbf{P} = \mathbf{NP}$, which in turn would be a million-dollar groundbreaking discovery. A much more realistic and down-to-earth path is the one stated in (ii); as was hinted before, we cannot have a solution that is (a) optimal, (b) fast and (c) general [1], for hard computational problems. The most common way around this is to relax point (a) and *settle* for a "good enough" solution that can be obtained in polynomial time. An algorithm that produces such solution is called an *Approximation Algorithm* (AA).

Even though AAs produce sub-optimal solutions to the problems they are designed to solve, they provide theoretical guarantees on the distance to the optimal solutions. This distance is formalised by the *approximation factor* α. Notice that the latter differences AAs from heuristics like Genetic Algorithms (GA) ([62]) and Tabu Search (TS) ([63]), as these methods, despite working generally well in practice, do not provide any guarantees for the solutions they compute.

Formally, we define approximation algorithms as Williamson *et al.* ([64], Ch. 1, p. 14):

***Definition* 1** (α-approximation algorithm)**.** *We call $\mathcal{A}$ an α-approximation algorithm for an optimisation problem if and only if $\mathcal{A}$ is a polynomial-time algorithm that, for all instances of the problem, produces a solution whose value is within a factor of α of the value of an optimal solution.*

[1] that is, a solution that works for all instances of the problem.

Definition 1 presents a concise summary of our discussion so far: AAs are *fast* algorithms that provide solutions that are always within an α factor of an optimal solution. Furthermore, this formal definition unveils two fundamental traits of AAs. First, AAs are problem dependent and the problem they are designed for is an *optimisation problem*; that is, a problem whose solution is found by *minimising* or *maximising* a specific mathematical function, *the objective function*, over a well-defined set of solutions, and possibly subject to a set of *constraints*. Second, the solutions produced by AAs are not only *for some* instances of the optimisation problem, instead, the approximated solutions hold *for all* instances in the solution space of the problem. This is a strong guarantee, and a powerful advantage of AAs. Hence, going back to the three desirable traits of an algorithmic solution *i.e.* optimality, speed and generality, approximation algorithms give fast and general solutions at the cost of relaxing optimality.

3.3 The Framework of Coresets

The framework of coresets is closely related to the concept of approximation algorithms. The method was born in the field of Computational Geometry as a systematic approach for finding the *minimum* number of input points that are necessary to approximate the solution of various geometric problems, most notably *shape fitting problems* ([65]). Even though the term "coreset" was coined in the work of Agarwal *et al.* ([23]), the idea was present in the research community under names such as "certificates" and ϵ-nets ([66], [23]).

A coreset is a small *compact* mathematical set that approximates a bigger set ([67]). As AAs, coresets provide a theoretical guarantee on the approximation quality via an approximation factor, for a specific optimisation problem of interest. Unlike AAs, coresets do not attempt to re-define any algorithmic solution in order to speedup computations; instead, they solely work on the *input data* for the problem at hand by attempting to reduce its size to a much smaller number. There are three important aspects to this reduction in the data size:

1. **It is controlled.** The reduction in size is not arbitrary in the sense that the *loss* of information is bounded by an approximation factor. Thus, without loss of generality, a coreset preserves the main properties of the set of data it approximates.

2. **It is fast.** The computation of a coreset, in principle, should be fast. Hence, an algorithm that reduces a set of data to a coreset, is expected to converge in polynomial time. This is another similarity coresets share with AAs; however, while AAs enforce this by definition, coresets do not. Hence, for coresets, this is an informal requirement which, still, has a very important effect in practical applications.

3. **It is problem dependent.** As its approximation algorithm counterpart, a coreset is a small set that approximates a bigger set of data for a specific optimisation problem. That is, since a coreset can be seen as an attempt to retain the most valuable pieces of information from a set of data, a well-defined notion of what is *valuable* is necessary.

The formal definition for coresets can be stated as follows: let function f be the objective function for some optimisation problem and $\mathcal{D}$ be the input data. Define a set of feasible solutions for f, Q; the members $q \in Q$ are generally referred as *queries* and Q itself is referred as the *query space* ([68]), because one can *query* the value of f for each $q \in Q$ by computing $f(\mathcal{D}, q)$. The set $\mathcal{C}$ is said to be a Δ-coreset for $\mathcal{D}$ with respect to f if the following condition holds for all $q \in Q$ [67]:

$$|f(\mathcal{D}, q) - f(\mathcal{C}, q)| \leq \Delta f(\mathcal{D}, q) \tag{3.1}$$

where $\Delta > 0$ accounts for the error incurred for evaluating f over $\mathcal{C}$ *i.e.* instead of using the full input data $\mathcal{D}$. This expression establishes the main error bound offered by coresets. Indeed, there is some information loss when the input data is replaced with its coreset, but this loss is bounded and controlled via the error parameter Δ. Notice that this error factor is analogous to the α factor in AAs; furthermore, as AAs, the error guarantee holds for all possible solutions of the

given optimisation problem *i.e.* for all queries in the query space Q. Finally, as was hinted before, it is expected that $|\mathcal{C}| \ll |\mathcal{D}|$ holds.

3.4 Related Work on Coresets for Machine Learning

As discussed on Section 2.2, machine learning programs attempt to find meaningful patterns in their input data. These patterns are crucial for obtaining a better understanding of the underlying data-generating mechanisms at play. Thus, in the field of machine learning, more input data is, in principle, and at the very least, desirable; this is because having more data available inevitably means better chances of unveiling more aspects of the *ground-truth* phenomenon of interest. Paradoxically, computer programs in general become slower as their input data grows; this is the big data paradox we have discussed in the previous chapter: on one hand, it is highly desirable to gather as much data as possible as this generally improves the output of learning programs , and on the other, the more data we collect, the more computationally exhaustive learning algorithms become.

In the last decades, coresets were introduced in machine learning in order to provide a middle-ground solution to the above dilemma for hard learning problems. Using the existing coreset theory, the machine learning community, alongside computer science and computational geometry theorists, have developed different approaches for constructing small coresets that approximate large input data. The most studied learning problem in this regard is clustering, both in its special cases such as k-Means ([67], [69], [70]) and k-Median ([70], [71], [44]), and in its generalisation to M-estimators ([68], [72]). An important number of coreset results for clustering can be found in [43] and [73].

There are also some incursions in the supervised-learning setting; specifically, there exists coreset constructions for the problem of linear regression, along with its constrained variants *i.e.* LASSO regression, Ridge regression and Elastic Net regression ([74]), and recently, some advances on coresets for neural networks were achieved ([75], [76], [77]). The most studied supervised-learning problem, however,

is that of Logistic Regression (LR), for which Huggins *et al.* designed a novel algorithm for summarising the input data based on existing coreset results for clustering ([78]); a follow-up work was done by Munteanu *et al.* ([79]) where a new complexity measure for the analysis of large data-sets was introduced, and new lower bounds for coresets could be derived with respect to this measure. Independently from these works, from an optimisation standpoint, Reddi *et al.* ([80]) proposed a theoretical iterative approach for compressing input data assuming an *Empirical Risk Minimisation* (ERM) setting ([2], Ch. 2, p. 15) instantiated with the *logistic loss* and the *hinge loss* functions. Another supervised-learning problem that has enjoyed the attention of the coreset community is that of Support Vector Machines (SVM) ([81]); this followed the groundbreaking result in computational geometry by Bădoiu and Clarkson [82] that showed that the $\ell 2$-SVM problem can be reformulated as the problem of finding the Minimum Enclosing Ball (MEB), and in this case, the coreset includes the points lying furthest away from the current centre of the current MEB. Many notable coreset algorithms for compressing input data for SVM followed directly from this result, this includes the Core Vector Machine (CVM) ([83]), its simplified version, named Ball Vector Machine (BVM) ([84]), and the recent Approximation Vector Machine (AVM) ([35]).

3.5 Coreset Construction

The expression in (3.1) states that evaluating function f over a Δ-coreset $\mathcal{C}$, which was obtained from the input data $\mathcal{D}$, gives an output value that, in the worst case, can deviate by a Δ factor from *any* solution in the query space. A natural question to ask is how to construct, or find, a set $\mathcal{C}$ that fulfils this requirement.

It turns out that the answer to the above question is far from trivial. First, the answer depends heavily on f: a set of points that fulfils the requirement in (3.1), for example, for the problem of k-Means will most likely not work for the problem of Logistic Regression as solving these problems involve different settings and different objective functions (see Section 2.3.1 and Section 2.4.1). In some sense, this can be seen as a consequence of the famous impossibility result known

as the No-free-lunch (NFL) theorem ([85]). Second, even when a specific problem has been fixed, an algorithm for selecting coreset points must be designed and a theorem stating that such algorithm produces a set of points that approximates the original set of points, as shown in (3.1), has to be proved.

The most important approaches for constructing a Δ-coreset for ML problems inevitably fall into one of the below four categories. As before, assume that $\mathcal{D}$ is the input data and that $\mathcal{C}$ is a Δ-coreset for $\mathcal{D}$ with respect to function f.

1. **Geometric decomposition ([70]):** this category covers a set of procedures that attempt to decompose, or discretise, the d-dimensional space where the input points reside. Geometrically, a *grid* is drawn onto the input points and, for each *cell* of the grid, a *representative point* has to be chosen. Hence, by collecting all representative points, one obtains a Δ-coreset. This methodology was the first approach used to construct coresets, and it has proved to be effective for compressing data for many computational-geometry problems.

2. **Randomised selection ([71]):** this category refers to a well-established framework, generally based on geometrical arguments, that involves defining a sampling distribution to account for the *importance* of each input point. A naive procedure for compressing input data using randomised selection is to simply use an uniform distribution; however, albeit straightforward, uniform sampling is unable to produce small coresets for non-outlier-resistant problems ([86]). A more involved procedure is offered by the *Sensitivity Framework* proposed by Feldman *et al.* in [71], where a non-uniform distribution, concentrated on a real-valued measure called *the sensitivity*, is used for sampling *sensitive*, or important, points with respect to function f. Thus, each input point is assigned a *sensitivity score* which captures the point's contribution toward the objective function f. Algorithms following this line of work are generally parameterised by a *probability of failure* δ. This is the probability that the guarantee in Equation 3.1 does not hold after the compression process.

3. **Iterative selection ([82]):** this set of strategies aims to construct coresets by using tools and analysis from convex optimisation. Most of the techniques under this category use coresets implicitly while optimising function f. In fact, even the well-known sub-gradient method ([87]) can be seen as an iterative algorithm that optimises function f over a coreset ([82]). The interested reader can find more details on this approach in [88], [89], [90] and [91].

4. **Projections ([92]):** this category involves techniques that are largely inspired by the Johnson-Lindenstrauss lemma ([93]), a result that states that the input data can be embedded in a much lower-dimensional space such that the distances among input points are *approximately* preserved. Thus, a coreset constructed by projecting input points onto lower dimensional spaces generally preserve the number of input points *i.e.* $|\mathcal{C}| \leq |\mathcal{D}|$, but since the coreset points live in a lower-dimensional sub-space, learning programs do converge faster. Projection-based coresets are hence well-suited for high-dimensional input data as many learning problems become simply intractable as the number of dimensions increases, this is commonly known as the *curse of dimensionality* ([29], Ch. 1, p. 33).

From these four approaches to construct coresets, the first two are fundamental to the rest of this thesis. The advantage of using geometric decomposition or randomised selection for obtaining a coreset is that the obtained coreset $\mathcal{C}$ is, in those cases, a *subset* of the input data $\mathcal{D}$, which means that the input space is preserved after the compression process. Projections and iterative approaches, however, often apply mathematical transformations *e.g.* linear transformations, to the points in the input data in order to obtain the coreset points. These transformations generally project the input space into different sub-spaces and thus, the input points and the coreset points cannot be associated after the compression is done. Furthermore, in the latter cases, the coreset is a completely new dataset and hence the prediction results obtained from learning over this coreset are hard to interpret in terms of the original input data $\mathcal{D}$.

The next section presents two algorithms that offer proven guarantees for computing a coreset for the classification problems defined in Section 2.3. The first one is the *Approximation Vector Machines* (AVM) algorithm ([35]), and it exploits the fact that the SVM objective function in Equation (2.5) can be reduced to the problem of finding the Minimum Enclosing Ball (MEB) for a set of d-dimensional points. This fact allows the use of well-known discretisation-based ideas for constructing coresets. The second algorithm is named *Randomised Coreset Algorithm* (RCA) ([78]) and it is an approach that relies on non-uniform sampling for constructing a coreset for the LR problem. RCA performs *importance sampling* in order to select input points with meaningful contribution to the LR objective function, defined in Equation (2.4).

3.5.1 Deterministic Coreset for SVM

The method of Approximation Vector Machines, AVM for short, defines a theoretically proven approach for compressing the input data into a coreset when the objective function at hand is the one for soft-SVM. AVM has two interesting features: first, it proposes a deterministic approach to coresets and thus there is no probability of failure (δ) in the compression process. The probability of failure cannot be omitted in randomised algorithms. Second, AVM does not only compress the input data, but also iteratively constructs a SVM model, $\bar{\theta}$, during the compression process. Therefore, after AVM converges, we obtain both: a fully trained SVM model [2], and the coreset over which the returned model was trained. This style of output is indeed uncommon in the coreset literature as most coreset algorithms output only the obtained coreset. This particular aspect of AVM served as inspiration for one of our main contributions of this thesis, the RDSF method, to be presented in Chapter 6.

Returning to our AVM presentation, it becomes clear from our discussion so far that AVM does apply some *solving* strategy for the SVM objective function during

[2] notice that "fully trained" means that no more training is needed after AVM converges. This obtained model, of course, is sub-optimal as it was constructed from a coreset, not from the full input data.

its execution. Indeed, AVM can be described as a coreset-based implementation of the well-known *Stochastic Gradient Descent* (SGD) ([94]) solver. In other words, what Le *et al.* proposed in [35] is to exploit the sequential nature of SGD to incrementally build a SVM model $\bar{\theta}_t$, $\forall t \in (1, \ldots, T)$, over *representative input points*, where T is the total number of sequential updates performed to the model. Furthermore, Le *et al.* proved that, after T iterations, the final AVM solution, call it $\bar{\theta}_T$, is not far from the optimal SVM solution, θ^*.

Before providing a full description for AVM, it will be instructive to describe the standard SGD steps. At each iteration t, SGD performs the following operations:

1. choose an example from the input data *i.e.* a tuple (x_i, y_i) from $\mathcal{D}$.

2. set the learning rate η_t *e.g.* $\eta_t = 1/\lambda t$ (as recommended in [38]), where λ is the regularisation parameter in the SVM objective function (2.5).

3. set the update for t as $g_t = \lambda\theta_t + \nabla\ell(\theta_t; x_i, y_i)$, where $\nabla\ell(\theta_t; x_i, y_i)$ is the gradient of the loss function ℓ *i.e.* the hinge loss presented in Section 2.3.2, evaluated at (x_i, y_i).

4. perform the update $\theta_{t+1} = \theta_t - \eta_t g_t$.

Concentrating on step 3 above, notice that for the hinge loss it is true that the gradient can be represented as $\nabla\ell(\theta_t; x_i, y_i) := \alpha_t\phi(x_i)$, where α_t is a scalar and ϕ is our mapping to the feature space. These are the support vectors and their α coefficients discussed on Section 2.3.2. If $\alpha_t \neq 0$ then x_i is said to be a support vector; and furthermore, the SGD solution at iteration t can be expressed in terms of those support vectors, namely, $\theta_t := \sum_{i=1}^{t} \alpha_i^{(t)}\phi(x_i)$. In practice, one would want to set $t = T = n$, where n is the total number of training examples.

We have mentioned in our discussions on SVM that the number of support vectors grows very quickly, especially when we use kernels. AVM allows us to mitigate the curse of kernelisation by only increasing the model size when the newly seen point is *far* from previously processed input points.

We are now in a good position for defining what representative points means for AVM. To do that, we need the following definition: given a domain space X *e.g.* our object space, a δ-coverage for X is defined as below.

***Definition* 2** (δ-coverage)**.** *The collection of sets $\mathcal{P} = (P_i)_{i \in I}$ is a δ-coverage of X if and only if $X \subset \cup_{i \in I} P_i$ and $D(P_i) \leq \delta \, \forall i \in I$, where I is the index set and $D(P_i)$ is the diameter of the set P_i i.e. the maximal pairwise distance between any two points in the set. Furthermore, each element $P_i \in \mathcal{P}$ is referred as a* cell.

To put the above definition in simpler language, a δ-coverage, called above $\mathcal{P}$, is a partition of a set of points into cells or containers, each having a diameter of *at most* δ. Hence, to AVM, all the points in the same cell are approximately the same, and thus all of them can be represented by a single point, called the *core point*. In theory, *any* point in a cell can be chosen as its core point. In practice, due to its sequential nature, AVM needs to construct the δ-coverage for the input data *on the fly* and hence it deterministically chooses the centre of each cell as its core point. The algorithm to construct the δ-coverage is described in Algorithm 11. Notice that it discretises the input space into *hyperspheres* or *balls*, each of them containing its own core point. Each cell P_i is a *ball* which is constructed with an input point as a centre and with fixed radius of $\delta/2$. Notice further that the first input point seen always becomes a centre, and hence a core point (c_1). If the distance of the second input point to the first one, which is a core point, is greater than $\delta/2$, then this point also becomes a core point (c_2) and is used as the centre for a second ball, and so on. Finally, the coreset is the union of all the core points $\mathcal{C} := \cup_{i \in I} \{c_i\}$, where each c_i is the centre of a ball $P_i \in \mathcal{P}$.

To further build intuition, Figure 3.1 shows the output of this algorithm for 230 synthetic 3-dimensional samples. The coreset is the set containing only the centres of the balls. Notice that this directly implies that the coreset size is indeed the number of enclosing balls used to discretise the input space.

Equipped with all of the above discussion, we can appreciate the full AVM method in Algorithm 10. It is easy to see the SGD structure in the algorithm;

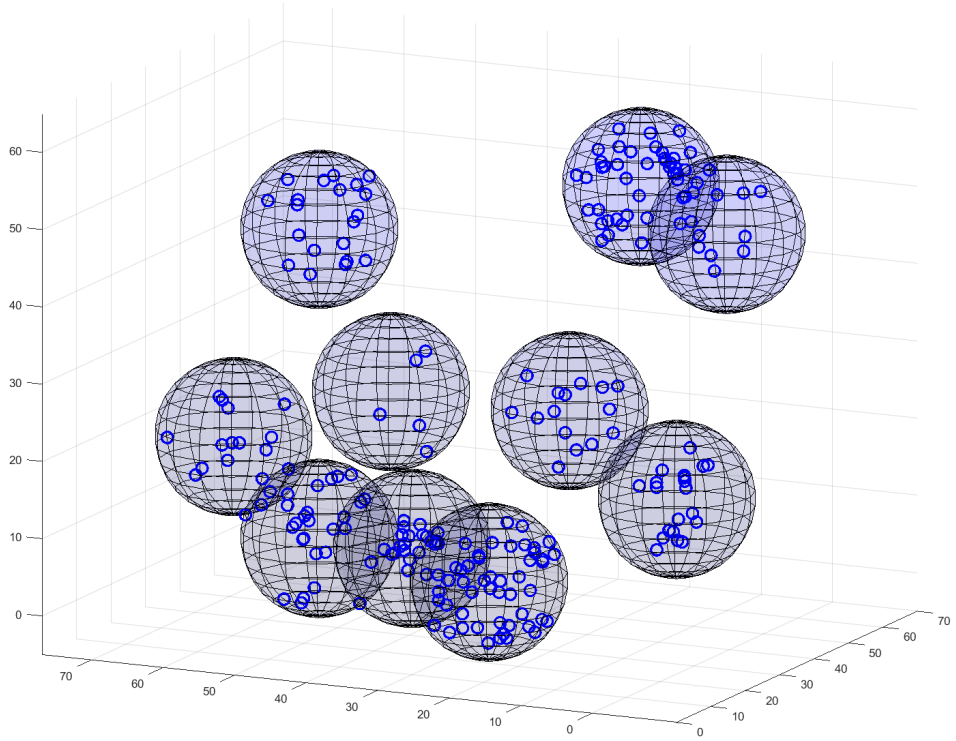


Figure 3.1: An illustration of the 3D hyperspheres coverage for 230 samples (the blue dots) from a synthetic dataset. All samples in each ball may be represented by its core point: the centre of the ball, which reduces the total number of samples from 230 to just 10.

however, notice that now we fetch the core point representing the cell in which the current input point resides, and perform a traditional SGD step using it. AVM guarantees that if we proceed that way, we can obtain a solution $\bar{\theta}_T$ such that $\mathbb{E}[|f(\bar{\theta}_T) - f(\theta^*)|] \leq \epsilon$, which in expectation gives us the main theoretical coreset guarantee, as defined in Equation (3.1). For the theoretical proof on AVM's convergence we refer the reader to [35], pages 10-12.

One very important property of the AVM definition is that it does not provide a way to control the number of core points. This is a very peculiar feature as most coreset constructions use a special parameter for this. In practice, this should be handled carefully; in fact, to use AVM as part of our methods in the next chapters, we impose a *budget size* for AVM in order to control the size of the coreset.

Algorithm 1: An algorithm for constructing $\mathcal{P}$, as defined in [35]. Each P_i is a ball with radius $\delta/2$.

$\mathcal{P} \leftarrow \emptyset$
$M \leftarrow 0$
for $i \in 1, 2, \ldots$ **do**
  receive a (x_i, y_i) pair
  $j_i = \arg\min_{k \leq M} ||x_i - c_k||$
  if $||x_i - c_{ji}|| \geq \delta/2$ **then**
    $M = M + 1$
    $c_M \leftarrow x_i$ `// Core point`
    $\mathcal{P} = \mathcal{P} \cup [\mathcal{B}(c_M, \delta/2)]$ `// Construct` P_i
  end
end

Algorithm 2: The AVM algorithm proposed by Le *et al.* Notice that the SGD update rule is expressed in terms of α_t and $\phi(\cdot)$. The model θ is constructed only from the core points.

Input: $\mathcal{D}$: input data, λ: regularisation parameter, $\mathcal{P} := (P_i)_{i \in I}$: δ-coverage
Output: θ: SVM Model
initialise
$\theta_1 \leftarrow 0$
for $t \in 1, \ldots, T$ **do**
  get (x_t, y_t)
  $\eta_t \leftarrow 1/\lambda t$
  Find $i_t \in I$ such that $x_t \in P_{it}$
  $g_t = \lambda\theta_t + \alpha_t \phi(c_{it})$
  $\theta_{t+1} = \theta_t - \eta_t g_t$
end
return θ_{T+1}

3.5.2 Randomised Coreset for LR

This section presents the state-of-the-art method available for constructing a coreset for the problem of logistic regression. The RCA method, proposed by Huggins *et al.* , poses the task of constructing a coreset as the task of defining a *non-uniform distribution* for the input points, and then sampling from that distribution. We start to build intuition with the following general question: "why not simply use uniform random sampling?"

Indeed, the above question does lead us to coresets. Uniform Random Sampling (URS) can be seen as a naive randomised approach to compute a coreset. Consider

we have our input data $\mathcal{D}$ and that we want to reduce its size in order to speed up the computation of a machine learning algorithm. We can simply assign probability $1/|\mathcal{D}|$ to each example in $\mathcal{D}$ and then sample according to this distribution. The time needed to reduce the size of the input data in this way is virtually negligible. The problem with this simple approach, however, is that we have to sample a very large number of points to ensure low error ([86]); that is, in order to keep the discrepancy between the result obtained from the full input data and the sample bounded, an impractically large URS sample from $\mathcal{D}$ will be needed. This happens because different points make different contributions to the objective function at hand and, consequently, by assuming that all input points contribute equally to the objective function, important points could be left out of the coreset if the uniform sample is not sufficiently large. This result, however, does put us on the right direction.

RCA is guaranteed to produce a Δ-coreset of size M with probability $1-\delta$ for logistic regression. Thus, based on the objective function defined for logistic regression in Equation (2.3), RCA outputs a coreset for which the following theoretical guarantee holds:

$$|\mathcal{L}_n(\theta|\mathcal{D}) - \tilde{\mathcal{L}}_M(\theta|\mathcal{C})| \leq \Delta|\mathcal{L}_n(\theta|\mathcal{D})| \tag{3.2}$$

for all $\theta \in \Theta$, where $\mathcal{L}_n(\theta|\mathcal{D})$ is the negative of the log-likelihood function over the input data and $\mathcal{L}_M(\theta|\mathcal{C})$ is the *weighted* negative of the log-likelihood function evaluated over the Δ-coreset $\mathcal{C}$, with $|\mathcal{C}| = M$. Notice that the coreset guarantee holds for the whole parameter space where θ lives, $\Theta \subseteq \mathbb{R}^{d+1}$.

Unlike the uniform-sampling approach where equal sampling probabilities are assigned to all the elements in $\mathcal{D}$, RCA defines a non-uniform distribution that attempts to capture *how important* each input point is with respect to the log-likelihood of the full input data $\mathcal{D}$. This importance is materialised in a score called the *sensitivity* of the point, which is the central quantity for constructing coresets non-uniformly. Once the sensitivity is computed for each input point, M points are sampled according to the sensitivity scores. The final step is to compute the *weights* for each sampled point, which are generally inverse-proportional to the

Algorithm 3: RCA: a randomised algorithm to construct coresets for Logistic Regression.

Input: $\mathcal{D}$: input data, $Q_{\mathcal{D}}$: k-clustering of $\mathcal{D}$ with $|Q_{\mathcal{D}}| := k$, M: coreset size, R : radius of bounding Euclidean ball
Output: Δ-coreset $\mathcal{C}$ with $|\mathcal{C}| = M$

initialise;
for $i = 1, 2, ..., n$ **do**
  $m_i \leftarrow$ `Sensitivity`$(n, Q_{\mathcal{D}}, R)$; `// Compute the sensitivity of each point`
end
$\bar{m}_n \leftarrow \frac{1}{n}\sum_{i=1}^{n} m_n$;
for $i = 1, 2, ..., n$ **do**
  $p_i = \frac{m_i}{n\bar{m}_n}$; `// compute importance weight for each point`
end
$(K_1, K_2, ..., K_n) \sim$ `Multi`$(M, (p_n)_{i=1}^{n})$; `// sample coreset points`
for $i = 1, 2, ..., n$ **do**
  $w_i \leftarrow \frac{K_i}{p_i M}$; `// calculate the weight for each coreset point`
end
$\mathcal{C} \leftarrow \{(w_i, x_i, y_i) | w_i > 0\}$;
return $\mathcal{C}$

sensitivity scores, and return the M weighted points. RCA is guaranteed to produce a Δ-coreset of size M with probability $1 - \delta$ for logistic regression.

RCA does produce coresets with better approximation guarantees compared to uniform sampling. The downside is, however, that defining and computing the sensitivities is very challenging because (i) computing the exact sensitivity of each input point is not computationally tractable ([78]); thus, lower and upper bounds for it need to be systematically defined; then, a proof to show that these bounds yield to small coresets is necessary; (ii) the bounds should be efficiently computable. On this front, it is worth mentioning that a coreset algorithm needs to inspect each input point at least once. Hence, by "efficiently computable", we mean linear time in the number of input points n.

It is important to mention that Huggins *et al.* followed the well-established *sensitivity framework* [71] to design their algorithm. This framework was successfully used to construct coresets for different instances of clustering problems such as k-Means and k-Median. The main idea is to formulate the coreset construction

as the problem of finding an ϵ-approximation [95] , which can be computed using non-uniform sampling based on the importance of each data point, in some well-defined sense [3]. Hence, each example in the input data is assigned an importance score, and then a sampling based on the distribution of those importance scores is performed in order to select the examples that will be included in the coreset.

An approximation to the optimal clustering of the input data is required in order to calculate such importance scores. For each point (example), the sensitivity score is computed by taking into account the distance between the point and its nearest (sub-optimal) cluster centre obtained from the approximation. The next step is to sample M points from the distribution defined by the sensitivity scores, where M is the size of the coreset. Finally, each of the M points in the coreset is assigned a positive real-valued *weight* which is the inverse of the point's sensitivity score. The sensitivity framework returns a coreset consisting of M weighted points. The theoretical proofs and details can be found in [71].

If we inspect Algorithm 3 closely, we can see that it is assumed we have some k-clustering of the input data, Q, where each $q \in Q$ is a cluster centre in the input data $\mathcal{D}$ and $|Q| = k$ (See Section 2.4.1 for a full definition of k-Means clustering, for example). These centres are used to compute the sensitivity scores (lines 2-4); then, sensitivities are normalised and points get sampled (line 5); finally, the weights, which are inverse proportional to the sensitivities, are computed for each of the sampled points (lines 6-12). Thus, even though the obtained coreset is for logistic regression, RCA still needs a clustering of the input data as it is common for any coreset algorithms designed using the sensitivity framework.

Theoretically speaking, RCA uses a k-clustering of the input data to obtain a bound on the sensitivities. The algorithm also receives a parameter R: a real-valued quantity that bounds the parameter space, $\mathbb{R}^{d+1}$, to an Euclidean ball of radius R. As mentioned early, the coreset guarantee in Formula (3.2) 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

[3] for our discussion, it is enough to state that this importance is a real-valued score in the half-open interval $[0, \infty)$

proofs and technical details on how/why this clustering-based bound is indeed a sensitivity bound for logistic regression is out of the scope of this discussion; all these details can be found in [78].

Practically speaking, the use of k-clustering 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 ([96], [78]). 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, RCA produces different coresets at each run.

Remark 1. *In the description of Algorithm 3, we hide the coreset dependency on the coreset error parameter Δ. There is a good reason for doing this. When theoretically designing a coreset algorithm for some fixed problem, there are two error parameters involved: $\Delta \in [0, 1]$, the "loss" incurred by coresets, and $\delta \in (0, 1)$, the probability that the algorithm will fail to construct a coreset. Then, it is necessary to define the minimum coreset size M in terms of these error parameters. The norm is to prove there exists a function $t : [0, 1] \times (0, 1) \to \mathbb{Z}^+$, with $\mathbb{Z}^+$ being the set of all positive integers, that gives the corresponding coreset size for all possible error values i.e. $t(\Delta_1, \delta_1) := M_1$ implies that M_1 is the minimum number of points needed in the coreset for achieving, with probability $1 - \delta_1$, the classical coreset guarantee that instantiated for logistic regression in inequality (3.2), for Δ_1. However, in practice, one does not worry about explicitly giving the error parameters as inputs; since each coreset algorithm comes with its own definition of t, one only needs to give the desired coreset size M and the error parameters can be computed using t. Finally, t defines a fundamental trade-off for coresets: the smaller the error parameters, the bigger the resulting coreset size i.e. smaller coresets may potentially lose more information than bigger coresets. For RCA, Huggins et al. proved that*

$t(\Delta, \delta) := \lceil \frac{c\bar{m}_n}{\Delta^2}[(D+1)log\,\bar{m}_n + log(\frac{1}{\delta})] \rceil$, *where* D *is the number of features in the input data,* $\bar{m}_n$ *is the average sensitivity computed over all input examples and* c *is a constant. The mentioned trade-off can be appreciated in the definition of* t*.*

Figure 3.2 shows a single run of RCA over 100 two-dimensional points which were synthetically generated to illustrate how the coreset algorithm works in practice. The algorithm uses the clustering of the data to define the sensitivity for each point. Then, it samples according to the *non-uniform* distribution of the sensitivity. Finally, it computes the weights and returns a coreset. The only thing left to do in this case is to run a LR solver over the coloured points in plot (d); using conformal prediction language, the weighted coreset can be used as a proper training set for training a single-point LR predictor. Notice that to compute the coreset, RCA only needs the example's object x. The label y is only used in the learning phase [4].

To finish the RCA exposition, the following final observations regarding the algorithm can be noted: 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 becomes intuitive if one considers the fact that RCA is 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 3.2; 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 $\bar{m}_n$ (Remark 1): 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 3.2, 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

[4] Coreset points retain the same labelling.

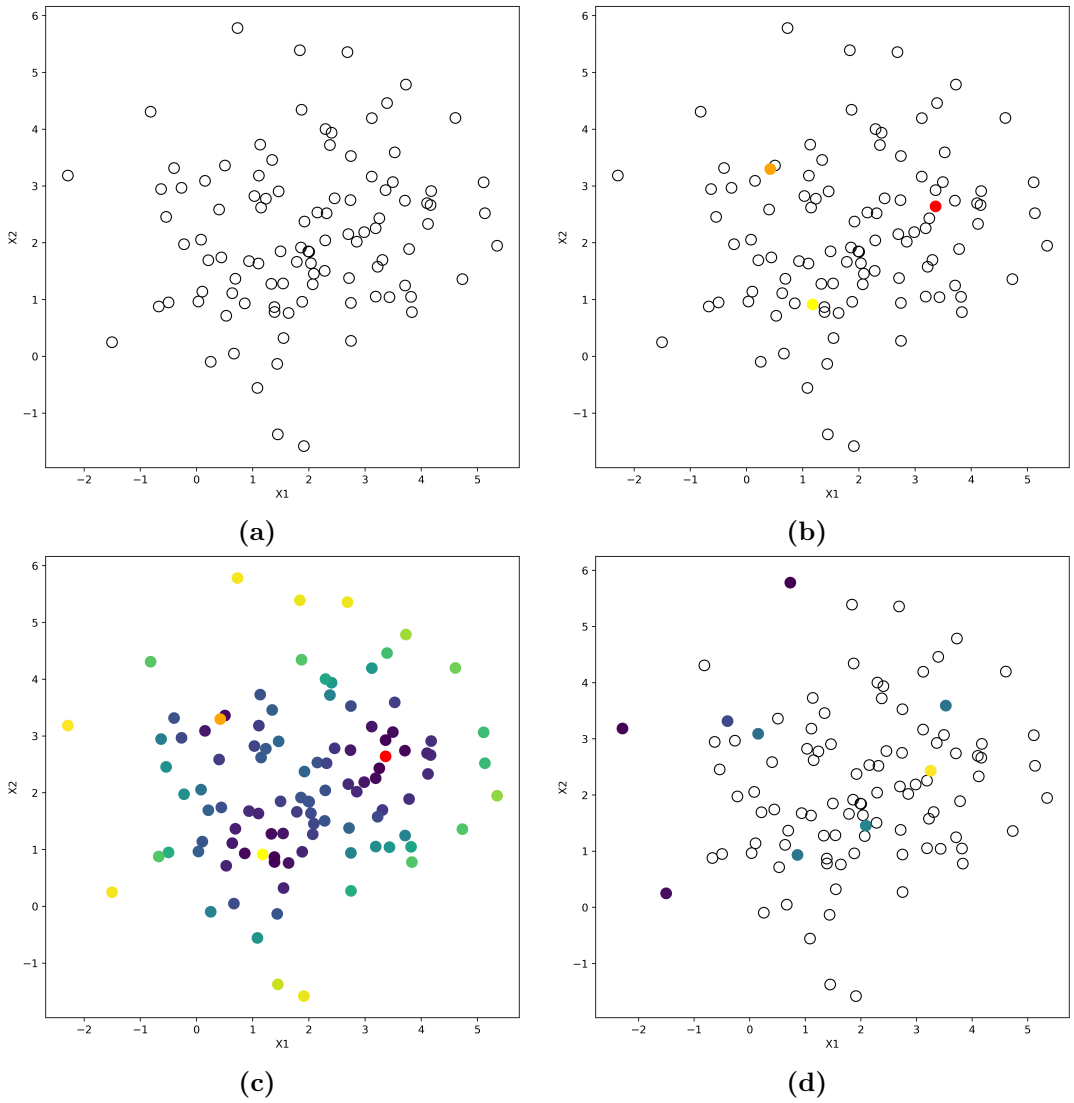


Figure 3.2: A step-by-step run of RCA over a simple 2-dimensional synthetic dataset; (a) shows an illustrative synthetic 2-dimensional data for which a logistic regression coreset is computed using RCA; the data has 100 points; (b) displays a k-clustering of the data to be used by the coreset algorithm 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.

into a very bright colour. Hence, when the sensitivities are too high, we virtually degenerate into uniform sampling; 4) RCA 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, RCA is a quite efficient coreset algorithm and, as the size of data gets very large,

its gains in running time become quite useful.

3.6 The Size and Error Trade-off

A key point when discussing algorithms to construct coresets is the inevitable inverse relationship between the size of the generated coreset and its error. As our formal definition states, a coreset approximates the input data up to a factor Δ. Even though every coreset algorithm provides this theoretical guarantee, the size of the resulting coreset, typically expressed as a function of Δ, can be very different from algorithm to algorithm. Table 3.1 shows what the coreset sizes look like for the well-studied problem of k-Means clustering, which is the most investigated computational problem in the coreset community due to its geometrical interpretation. The table also shows the coreset sizes for the LR and SVM problems, as obtained by the RCA and AVM methods, respectively.

Table 3.1: The size-error trade-off of coreset algorithms for a set of ML problems. n is the size of the input data, d is the number of dimensions in the input data, Δ is the approximation error and k is the parameter for the number of clusters for the k-Means problem; $\mathcal{P}$ is the δ-coverage used in AVM.

Problem	**Coreset Size**	**Authors**
k-Means	$O(k\Delta^{-d}\log^{d+2} n)$	Har-Peled & Mazundar ([70])
k-Means	$O(k^3\Delta^{-(d+1)}\log^{d+2} n)$	Har-Peled & Kushal ([97])
k-Means	$O(dk\Delta^{-2}\log^8 n)$	Chen ([44])
k-Means	$O(dk\log k\Delta^{-4}\log^5 n)$	Feldman & Langberg ([71])
k-Means	$\tilde{O}((\frac{d}{\Delta})^{\mathcal{O}(d)}k\log^{\mathcal{O}(d)} n)$	Ackerman *et al.* ([98])
LR	$O(\frac{1}{\Delta^2}(d+1))$	Huggins *et al.* ([78])
SVM	$O(\lvert\mathcal{P}\rvert)$	Le *et al.* ([35])

As shown in the above results, it is often the case that coresets are order of magnitudes smaller than the input data size n *i.e.* the coreset size is logarithmic or near-logarithmic in n. Table 3.1 also sheds light on the competing forces of coreset error and coreset size: most of the coreset sizes have a polynomial dependence of $\frac{1}{\Delta}$ on the approximation error; that is, in most of the bounds shown above, the greater

the approximation error gets, the smaller the coreset size becomes, and vice versa. Notice that AVM is a special case because, as we mentioned earlier in the chapter, the method does not provide a bound on the coreset size explicitly. Thus, the maximum coreset size is the total number of enclosing balls that we have in the δ-coverage $\mathcal{P}$.

The main idea of Table 3.1 is to show how coresets react to the big data paradox discussed previously, for different machine learning problems.

3.7 Adequate Data Representation with Coresets

One natural question that often arises when using coresets in practical applications is the following: " How adequately can a coreset represent a data-set for a specific learning problem X?". The answer to such question relies, ultimately, on the machine learning practitioner. There are two important aspects that need to be discussed to shed light on this issue.

3.7.1 The theoretical aspect

Coreset algorithms, such as the ones presented in this chapter, offer a theoretical proof that states that when using the coreset as a surrogate for the original input data one possesses, the discrepancy in performance will be measurable and tractable with an error parameter Δ; and as the previous discussion in this section highlighted, the value of this error parameter will influence the final size of the coreset. As the chapter mentioned in Section 3.5.2 (Remark 1), each coreset algorithm comes with a theoretical bound for the coreset size, and such bound is written with respect to the coreset error parameter. For instance, the coreset size bound given by RCA can be written as $t(\Delta, \delta) := M := \lceil \frac{c\bar{m}_n}{\Delta^2}[(D+1)log\bar{m}_n + log(\frac{1}{\delta})] \rceil$, where M is the coreset size, D is the number of features in the input data, $\bar{m}_n$ is the average sensitivity computed over all input examples and c is a constant. In essence, once one has a coreset algorithm for problem X, the coreset theory *guarantees* that the optimal solution found in the coreset will never deviate from the optimal solution in the original data-set by more than a factor of Δ (see [78] for a condensed exposition of the theoretical proofs involved when designing a coreset algorithm for logistic regression).

In similar fashion, Chapter 4 will show how the framework of Conformal Prediction also *automatically* produces a calibrated number of prediction errors using machine learning classifiers.

3.7.2 The practical aspect

Coreset theory automatically guarantees the tractability of the error induced by the data compression process. The theory, however, does not tell the user what error values to use for generating a coreset as this heavily depends on the application and setting the ML practitioner is interested in. To make this clear, suppose we have two scenarios: *scenario A* and *scenario B*, both exploiting the benefits of coresets in different fashion.

Scenario A

Scenario A consists in an industrial application of chemical compounds interactions detection with the goal of releasing a new drug that aims to reduce the probability of having a heart attack. In this setting, a very large training set containing records of different drugs' characteristics is available and the data scientist in charge wants to perform a *fast* classification using LR to test how well the interaction of this new drug can be predicted for some test data she has. Since her computational power *is modest*, she decides to use the RCA algorithm, presented in Section 3.5.2, to obtain a coreset that she can use *as a proxy* to the full input data available. Assume that the original training set has a total number of $2,000,000$ data points, each point having 70 dimensions, and that this data-set has an average sensitivity score of 2 [5] *i.e.* $n = 2,000,000$, $D = 70$ and $\bar{m}_n = 2$. Given the sensible nature of this application, the practitioner needs a coreset that allows for a very small error *i.e.* because of the implications of missing some drugs interactions may have on the users' health. Thus, she sets the error parameter for the coreset to $\Delta = 0.01$ and the probability error to $\delta = 0.01$. Plugging in *all* these parameters to RCA's coreset size function $t(\Delta, \delta)$, she obtains a coreset size of $M = 1,076,373$, which

[5] The average sensitivity score depends on the given input data, and it is calculated in the RCA algorithm using a k-clustering of the input data. See Algorithm 3

requires her to use slightly over 50% of the total amount of data available. This means that in the *worst case scenario*, *for all* possible solutions for the LR problem in the original data-set, the LR solution found in the coreset will be deviated from that found in the original drugs interactions data-set by a factor of 0.01.

Scenario B

Scenario B presents a different situation from the one discussed above. Here, the application at hand uses ML to give instant feedback to banking customers regarding their loan applications in automated fashion. To do this, the data scientist has a large database containing the banking activities of all the bank's customers from all the different branches. Since the nature of the application favours speed over precision, he decides to just look at a subset of the available data to decide whether the customer has good chances of getting the requested loan approved. The computational resources assigned to this application are modest, and the data scientist is only allowed to use 1Gb of data to make the decision for the loan requests. The software output is just for the customers to know whether their loan application is strong or not; at the end, the bank will have to get some further input from the customers to make a final decision. Thus, the ML practitioner decides to rely on a LR classifier to make the decision, and furthermore, she decides to use a coreset, instead of uniform random sample, of the customers database for giving the customer the requested feedback. She calculates that she can safely fit $1,000,000$ data points in the assigned 1Gb space, out of the $20,000,000$ data points stored in the database. Each of these input points has a number of dimensions of $D = 100$. Thus, she wants to calculate the *worst case error* for a coreset of size $M = 1,000,000$. She further sets the probability-of-failure parameter to $\delta = 0.01$. Assuming that the average sensitivity of the bank customers data-set is $\bar{m}_n = 50$ and that the coreset algorithm is RCA, putting these values into function $t^{-1}(M, \delta)$ [6] gives a coreset error bound of $\Delta = 0.3$. Hence, the approximate solution for LR found in the coreset can deviate, in the worst case, by 30% from the value of the optimal

[6] notice that this is the inverse function of t, and gives the coreset error with respect to the coreset size.

solution found in the full costumers database. In conclusion, a coreset can always track the worst case scenario for a specific problem. The *optimal parameter values* to construct a coreset, however, cannot be generalised for all possible applications.

3.8 Aggregations via Merge & Reduce

Finally, one important aspect of the coreset paradigm is that it allows the adaptation of batch coreset construction algorithm to distributed and streaming settings straightforwardly. This is a very attractive feature of coresets, and we include it here because the below properties will enable our contributions to be extended to those computational settings.

The coreset paradigm provides the following two properties ([68]):

- **Recursion:** coresets allow to be computed in recursive fashion, at the cost of *stacking up* the approximation error. Formally, if $\mathcal{C}_1$ is a Δ-coreset for $\mathcal{D}$, and $\mathcal{C}_2$ is a Δ-coreset for $\mathcal{C}_1$, then the recursion property asserts that $\mathcal{C}_2$ is a Δ^2-coreset for $\mathcal{D}$.

- **Addition:** two coresets can be safely merged together provided they are *representing* two non-overlapping sets of data. Formally, let $\mathcal{C}_1$ be a Δ-coreset for P_1 and $\mathcal{C}_2$ be a Δ-coreset for P_2, with $P_1 \cap P_2 := \emptyset$. Then, by the addition property, we have that $\mathcal{C}_1 \cup \mathcal{C}_2$ is a Δ-coreset for $P_1 \cup P_2$.

The combination of the Recursion and Addition properties gives a powerful protocol for adapting existing coreset construction to more complex settings such as the ones mentioned above. One could, for example, hold non-overlapping portions of some input data across many machines. In order to avoid transmitting large portion of data over the network, each machine could only transmit a coreset of its local data to a central server, which in turn can perform a straightforward *merge* operation (applying the addition property) to obtain a coreset representing all the data stored in the machines. Some works on this line can be found in works of Zhang *et al.* ([99]) and Feldman *et al.* ([100]).

Another computational scenario that becomes available as consequence of the two properties established above is that of *streaming*. In such setting, the input data arrives sequentially to our hypothetical machine, and the incoming influx of information is so big compared to the machine's storage capacity, that the input data can never be held in its entirety in memory. It turns out that coresets are quite powerful for tackling this challenging situation. The main technique is referred as *merge & reduce* and can be traced back to the 1980's work of Bentley *et al.* ([101]); it was in the work of Har-Peled and Mazumdar ([70]), however, that merge & reduce was used alongside coresets. The approach can be summarised in three steps: (i) from the virtually infinite sequence of incoming data points, wait until the first t points arrive and, once this happens, compute a coreset for those t points. Store the coreset and disregard the original input points; (ii) as soon as there are c coresets stored in memory, merge the coresets, via the addition property, and compute the coreset of the union of the c coresets, via the recursion property. Now there are coresets with different error guarantees *i.e.* the recursion property accumulates the error Δ multiplicatively. Thus, this naturally imposes a hierarchical structure on the coresets kept in storage with respect to their approximation error Δ; more concisely, that structure can be accurately represented with a tree with arity at most c; (iii) repeat steps (i) and (ii) to keep populating the coresets tree. An interesting observation of this technique is that the top-most coreset in the tree, is a coreset for the rest of the coresets in the tree. More details can be found in [70]. Also, a new approach, based on merge & reduce, was proposed by Braverman *et al.* ([68]).

3.9 Chapter Summary

Approximation algorithms are one of the most effective approaches to tackle hard computational problems in Computer Science. This idea largely inspired similar approaches in many mathematical fields and sub-fields. The paradigm of coresets is one of such approaches; designed originally to efficiently solve shape-fitting problems in computational geometry, coresets provide a systematic framework for provably correctly approximating sets of points. The technique has seen substantial

success in solving hard machine learning problems in the last decades, especially for the intractable problem of clustering. Coresets are small problem-dependent summaries of data. Once a coreset is computed, it can safely replace the original input data as it provide a guarantee, Δ, that the solutions found in the coreset will not be arbitrarily *far* from those found in the original input data. Thus, coresets do lose some information, but the loss is bounded. The construction of these small data summaries can be deterministic, randomised, iterative or based on projections. For our work, the former two will be of great importance. Therefore, we have presented two representative methods for those cases: the deterministic AVM and the randomised RCA.

The size of a coreset depends inversely on the approximation error; that is, smaller coresets allow for more information loss, and bigger coresets favour optimality as they restrict the error size further. Consequently, the best parameters for computing a coreset depend heavily on the application at hand.

The coreset paradigm is very attractive due to the properties of Recursion and Addition: the former allows one to compute the coreset of coresets, at the cost of accumulating error; the latter states that the union of coresets is also a coreset, provided that the merged coresets are summarising non-overlapping sets of data. It follows directly from these two properties that any solution to a computational problem that is based on coresets can be extended to distributed or streaming scenarios with little effort. The streaming setting, a particularly constrained computational model, is a natural fit for coresets via the merge & reduce approach, in which we can deal with an unbounded number of input points by merging and re-compressing coresets, building hence a tree containing coresets as leaves. Still, it is important to observe that coresets have limitations: they are entirely problem dependent and thus theoretical guarantees for one problem do not automatically transfer to any other problem. Furthermore, designing a coreset construction algorithm is a demanding task and there are many problems for which no coreset constructions are allowed.

In the next three chapters, we will propose a series of protocols to efficiently classify data, and the coreset technique will be the backbone for those protocols.

It is a profound and necessary truth that the deep things in science are not found because they are useful; they are found because it was possible to find them.

— J. Robert Oppenheimer
Atom and Void: Essays on Science and Community
([102])

4

Coreset-based Inductive Conformal Prediction

4.1 Introduction

Up to this point, we have presented two fundamental machine learning classifiers, LR and SVM; both of them allow us to predict labels for data we have not seen before. These predictions, however, have to be taken with a grain of salt as we cannot really know *how good* the predictions are until the correct labels are disclosed to us. In this Chapter, we present *Conformal Prediction* (CP) [13]: a framework where virtually any machine learning algorithm can be *plugged-in* in order to complement its output with *confidence* information. That is, CP answers not only the traditional question "what are the predictions for this set of objects?", but also the fundamental question of "how much can I trust these predictions?". These answers are given to us *before* we get to see the correct labelling of the data we are making predictions over. The output of CP for a specific object x, then, is not a predicted label $\hat{y}$ (single-point prediction), but a *set*, called *prediction set*, containing, for each possible labelling of the object *i.e.* $\forall y \in Y$, a *p-value* that quantifies the fraction of training examples that are at least as *different* from others as x is if we label it with y ([103]). The notion of *difference* here is grounded on a dissimilarity function called *non-conformity measure*, which needs to be defined *a*

priori. Furthermore, CP provides us with the following theoretical guarantee: if we disregard the labels whose p-values are smaller than a significance parameter ϵ, which is provided before running CP, then the correct label of the object at hand will be in the prediction set with probability at least $1 - \epsilon$.

The above powerful properties do not come for free, unfortunately. Due to its elevated computational overhead, the use of CP is mainly restricted to small datasets. This chapter addresses this issue with the algorithmic data-compression technique of coresets, presented in Chapter 3. Specifically, we will introduce the method of *Coreset-Based Inductive Conformal Prediction* (C-ICP), which replaces the training set in CP with a small coreset, obtaining substantial computational acceleration without altering the CP framework in any way. CP will be instantiated with the two classifiers introduced in Chapter 2: LR and SVM. Fundamentally, we will show that the theoretical properties of CP transfer naturally to C-ICP, and conclude that this first interaction between CP and coresets leave the door open for a new research line on CP with large data.

4.2 Conformal Prediction

We start the chapter relying on the machine learning notation introduced in Chapter 2. Assume the typical classification scenario where we want to classify some object, call it x_{n+1}, and further assume that this object has a (so far unknown) label y_{n+1} that can be one of the values in the set $\{1, -1\}$; the imminent learning question is *what is the correct label of object* x_{n+1}*?* To answer this classic question in learning theory, we would need to train a classifier on some objects sampled from the same space x_{n+1} lives; more concretely, we need to take a fixed number,n, of training objects, along with their respective labels, and construct a mathematical model that will allow us to predict a label $\hat{y}_{n+1}$ for the current object we are interested in classifying. This model is used to define a hypothesis, or single-point predictor, as we discussed in Section 2.3. Thus, the typical output of a learning algorithm in this case would be the predicted label $\hat{y}_{n+1}$ and *only* once the true label y_{n+1} becomes available to us we can know how well our classifier performed.

The main purpose of the framework of Conformal Prediction is to attempt to answer a second question in addition to the one stated above: *how good of a prediction is $\hat{y}_{n+1}$ for object x_{n+1}?*. To do this, CP *converts* single-point predictors into *conformal predictors*. In the above situation, CP would tell us how likely it is that $\hat{y}_{n+1} = 1$ and $\hat{y}_{n+1} = -1$. These values can be seen as reliability measures that become available to us before we get to see the true label for object x_{n+1}. This is why the term *Reliable Machine Learning* ([12]) is used when machine learning occurs through the CP framework. As mentioned before, CP receives an ϵ parameter, the *significance* parameter, which guarantees that if we predict a label whose reliability measure is greater than ϵ, then the average number of errors of CP will be *at most* ϵ ([104]). Hence, a conformal predictor does not output a *single-point* prediction for object x_{n+1} as, for example, the classifiers discussed in Chapter 2; instead, it gives us a *prediction set* consisting of all labels whose reliability measures are greater than the ϵ parameter. In the next section, we will formalise these ideas and we will also shed light on two fundamental properties of CP: *validity* and *efficiency*.

4.2.1 CP Notation and Assumptions

In order to properly define the CP framework, it is important to make use of the following compact notation. We define an *example* as $z_i := (x_i, y_i), \forall i$; that is, z_i is a tuple consisting of an object $x_i \in X$ and a its corresponding label $y_i \in Y$, where X is the *object space* and Y is the set of all possible labels, or the *label space*. We further set $X := \mathbb{R}^d$ and $Y := \{1, -1\}$. Therefore, the space where examples live can be mathematically written as $Z := X \times Y$.

As is usual in CP literature, we use a *bag* [1] as the main structure for holding our data, and it implicitly implies that the ordering of our data is of no importance for the output of conformal predictors ([13], p. 23). Thus, we define Z^n, a bag containing n examples, as $Z^n := \wr z_1, z_2, ..., z_n \int$, where $n \in \mathbb{N}$. As with all our previous definitions, we also define the *bag space*, the space containing all bags containing n examples, as Z^*.

[1] or a multiset.

It will be useful to also remind the reader of our functional definition for single-point predictors, as discussed in Section 2.3. Namely, we define a single-point predictor as a function $h : Z^* \times X \to Y$. Most machine learning predictors fall under this definition, including the LR and SVM presented in Chapter 2; that is, given a bunch of training examples $Z^n \in Z^*$ and a new test object $x \in X$, the single-point predictor h will output (predict) $\hat{y} \in Y$ as the label for x.

Finally, it is important to observe that the theory we are about to present relies on the Randomness assumption, stated in Section 2.3.

4.2.2 Non-conformity

Now that we have our notation and assumptions clear, we define a central concept for CP: *non-conformity*. Generally, a conformal predictor measures *how different* or *non-conformal* a new example is from a collection of examples we have already seen. This idea is formalised via the concept of a *non-conformity measure*. Formally, a nonconformity measure is defined as a function $\mathcal{A} : Z^* \times Z \to \mathbb{R}$, where the real-valued output of $\mathcal{A}$ is called a *non-conformity score*. Intuitively, this score tells us how different an example z is with respect to the examples in the bag Z^n. A crucial observation here is that the description for such function $\mathcal{A}$ usually relies on some learning model trained over the examples in Z^n; hence, this is where standard machine learning algorithms are *plugged-in* into the CP framework. As the work of Shafer and Vovk states in [103], virtually any single-point predictor can be used to define a non-conformity measure. Later in this chapter we will define non-conformity measures for LR and SVM; for now, however, it is not necessary to give a full definition of $\mathcal{A}$.

Using $\mathcal{A}$ as defined above, and considering the bag $Z^n := \wr z_1, z_2, ..., z_n \int$, a non-conformity score can be assigned to example z_i as follows:

$$\alpha_i := \mathcal{A}(\wr z_1, z_2, ..., z_{i-1}, z_{i+1}, ..., z_n \int, z_i) \tag{4.1}$$

Notice that in order to compute the real-valued non-conformity score for the example z_i, α_i, we exclude this example from Z^n; this is because measuring how non-conformal an example is compared to itself is trivial.

We can compute these scores for each member of Z^n and obtain a matching bag, $\wr\alpha_1, \alpha_2, ..., \alpha_n\wr$, containing all the non-conformity scores for each $z \in Z^n$. In the next section we will see that these values are of little interest to us in isolation from each other; however, by comparing them against each other, CP produces the p-values that can be used as informative reliability measures for our predictions.

4.2.3 P-values as Reliability Measures

With the main concepts in place, we can now fully define a conformal predictor. As before, we start from the functional definition. A conformal predictor can be defined as a function $\Gamma : Z^* \times Z \times (0,1) \rightarrow 2^Y$, where 2^Y is the set of all the subsets of Y and $(0,1)$ is the open interval between 0 and 1.

This function definition is fundamentally different from that of a single-point predictor in two major ways: first, the conformal predictor Γ outputs a *subset* of the label space Y, instead of a single element $y \in Y$; second, Γ receives a real-valued parameter, usually denoted as ϵ, that can take any value $0 < \epsilon < 1$; this is the significance parameter we have mentioned before, and it is used to regulate *the informativeness* of a conformal predictor.

We will expand our discussion on the above properties of Γ later in the Chapter; now, however, it is important to finish our formal description of a CP by using the non-conformity concepts introduced previously.

Let $\mathcal{A}$ be a non-conformity measure, Z^n be a bag containing n examples and x be an object whose label we want to predict. The first step is computing non-conformity scores for all the examples in Z^n and also for our new object x. Notice that this new object is an incomplete example in the sense that we have no knowledge about its true label, call it y_{true}; in fact, that is exactly what we are attempting to predict. Thus, we provide a *postulated* label to x ([104]), $y \in Y$, and then see how non-conformal the example (x, y) is compared to the examples in Z^n. Formally, for each $y \in Y$, we do the following non-conformity computation, as defined in Equation (4.1):

$$\alpha_i = \mathcal{A}(Z^n \setminus \wr z_i\wr \cup \wr(x,y)\wr, z_i) \;\; \forall z_i \in Z^n \tag{4.2}$$

Similarly, we compute the non-conformity score for the example (x, y), consisting of the new object and its postulated label:

$$\alpha = \mathcal{A}(Z^n, (x, y)) \tag{4.3}$$

Finally, we perform the following calculation using the non-conformity scores obtained in Equations (4.2) and (4.3):

$$p_y := \frac{|\{i = 1, 2, .., n : \alpha_i \geq \alpha\}| + 1}{n + 1} \tag{4.4}$$

where p_y is the *p-value* of the postulated label y.

As it has been hinted before, the non-conformity scores are only informative to us when we compare them against each other. This intuition is captured by the p-value p_y, which quantifies how non-conformal the object x is with respect to all the examples in Z^n if we postulate that its label is y. As done in [103], we will refer to Equation (4.4) as the *conformal algorithm*.

Thus, if p_y is small, we can conclude that when we label the object x with y, we obtain a very strange example, (x, y), with respect to the bag Z^n. On the other hand, if p_y is large, it means that the example (x, y) conforms well with the rest of the labelled objects in Z^n and hence we can conclude that y is a suitable label for x.

Following the conformal algorithm described above, we obtain a p-value for each label $y \in Y$. In order to produce the prediction set for object x, the conformal predictor Γ *rejects* labels with p-values less or equal to the significance parameter ϵ. We can formalise the prediction done by any conformal predictor as follows:

$$\Gamma^\epsilon(Z^n, x) := \{y \in Y : p_y > \epsilon\} \tag{4.5}$$

Notice that ϵ is now written on top of Γ, as is usual in conformal prediction ([12]). The expression in (4.5) formally shows that the conformal predictor will output as potential labels for x all the labels whose p-values are greater than ϵ. Hence, conformal predictors are generally called *set predictors* and, ideally, we want these sets to be as small as possible.

4.2.4 Validity and Efficiency

We have established that CP computes a sequence of non-conformity scores and, via the conformal algorithm, processes those scores into p-values, which give insights on the strangeness of the example being considered. It then outputs a set of possible labels for the object x, which we are interested in classifying. More concretely, the labels in the prediction set $\Gamma^{\epsilon}(Z^n, x)$ are the ones with p-values greater than the significance parameter ϵ. The fact that we have a set of predictions instead of a single-point prediction leads us to the concepts of *validity* and *efficiency*, both being closely-related properties of the prediction sets generated via CP.

We say that the prediction set $\Gamma^{\epsilon}(Z^n, x)$ is *valid* if and only if $\Gamma^{\epsilon}(Z^n, x)$ contains the correct label for x, y_{true}, with probability at least $1-\epsilon$ ([103]). The most powerful feature of conformal prediction is that *all* prediction sets that are generated via the conformal algorithm are, by default, valid. The proof of this claim can be found in [13], Chapter 2. Hence, validity is automatically provided by the CP framework; for instance, if we have a prediction set $\Gamma^{0.05}(Z^n, x)$, it is guaranteed that the correct label for x will be in the prediction set with 95% *confidence*. Furthermore, in this case, the maximum number of classification errors will be 5%. It is important to observe that this error bound is for the *average* number of mistakes and thus it should not be interpreted as a bound for each prediction made by the conformal predictor. It is also worth mentioning at this point that a classification mistake, in the conformal-prediction sense, occurs when the correct label of the object in question *e.g.* x, *is not* present in the prediction set $\Gamma^{\epsilon}(Z^n, x)$.

Notice that we have defined validity for the label prediction of a single object x. By the Randomness assumption stated in Section 2.3, it follows that the same guarantee will hold for any sequence of test objects we attempt to classify. Thus, given any sequence of t test objects $(x_1, x_2, ..., x_t)$ where $t \geq 1$, a conformal predictor $\gamma^{0.1}$ will misclassify at most 10% of the test objects, and the correct labels will be in the prediction sets 90% of the time.

A natural consequence of the concept of validity is that of efficiency. A prediction set output by a CP γ^{ϵ} is said to be *efficient* if the size of obtained the prediction

set is small. Given a prediction set $\Gamma^\epsilon(Z^n, x)$, the maximum efficiency possible for any prediction set is $|\Gamma^\epsilon(Z^n, x)| = 1$ *i.e.* there is only one label in the prediction set, and the minimum possible efficiency is $|\Gamma^\epsilon(Z^n, x)| = |Y|$ *i.e.* all the labels in the label space are in the prediction set.

The above two cases lie on both sides of the efficiency spectrum, and they are useful for building intuition on the role played by the significance parameter in CP. The first one, $|\Gamma^\epsilon(Z^n, x)| = 1$, is the ideal case; it means that the conformal predictor, at significance level ϵ, is $(1-\epsilon)\%$ sure that the only member in $\Gamma^\epsilon(Z^n, x)$ is the correct label for x. In general, we want to be as close to this situation as possible; in other words, we always look for significance levels that produce the most efficient prediction sets possible. On the other side of the spectrum, we have the case where $|\Gamma^\epsilon(Z^n, x)| = |Y|$; this can be interpreted, for this particular significance level, as the conformal predictor not being able to distinguish any meaningful pattern for x, given the information available to the conformal predictor *i.e.* the bag Z^n and also the non-conformity measure $\mathcal{A}$. Thus, in this case, Γ^ϵ trivially outputs the whole label space Y. However, this seemingly naive answer from the conformal predictor, sheds light on a fundamental property of efficiency: unlike validity, it is not automatically guaranteed by CP. In fact, we can see efficiency as the price to pay for having the strong guarantee of validity. Therefore, efficiency is the property of conformal prediction that is not automatically guaranteed.

Continuing with the instructive case where $|\Gamma^\epsilon(Z^n, x)| = |Y|$, it directly follows from the definition of prediction sets (Equation (4.5)) that every label $y \in Y$ has a p-value $p_y > \epsilon$. By increasing the value of the significance parameter, we can potentially obtain different prediction sets, with different sizes. This implies that, given two different significance values ϵ_1 and ϵ_2, where $\epsilon_1 < \epsilon_2$, we have that $\Gamma^{\epsilon_2}(Z^n, x) \subseteq \Gamma^{\epsilon_1}(Z^n, x)$. Hence, *nested* prediction sets are generated as we increase the significance level ([13], p. 8). The adequate significance value, of course, will be problem-dependent; however, as is implied by our definitions, the more nested the prediction set is *i.e.* the higher the significance level, the higher the chances that the predictor makes a mistake.

4.2.5 The Transductive Computational Setting

By inspecting the full definition of the CP framework, we can make two important observations: i) the CP framework is defined in sequential fashion; ii) computing non-conformity scores for n examples implies training at least n learning models.

These two issues are closely related. The theory of CP framework assumes an *online* setting in which, as new examples become available to us, the predictions become more accurate. Algorithm 4 presents the classic online CP setting. Notice that $\mathcal{E}_0^\epsilon$ is the total number of classification mistakes made by the algorithm at significance level ϵ.

Algorithm 4: Transductive Conformal Prediction Protocol.

Input: $\epsilon \in (0, 1)$: significance level

$\mathcal{E}_0^\epsilon \leftarrow 0$
for $n = 1, 2, 3...$ **do**
  Observe object x_n
  Compute prediction set $\Gamma_n^\epsilon \subseteq Y$
  Observe the label y_n
  if $y_n \in \Gamma_n^\epsilon$ **then**
    $err = 0$
  else
    $err = 1$
  end
  $\mathcal{E}_0^\epsilon = \mathcal{E}_0^\epsilon + err$
end

The protocol shown in Algorithm 4 is commonly known as *Transductive Conformal Prediction* (TCP) ([13]), where *transductive*, as indicated by Vapnik in [105], implies that learning and predicting are not separated processes, as is common in traditional *batch* machine learning procedures. Instead, these two processes are mixed together and they both happen for every new example that the algorithm sees.

TCP assumes an *online* setting where there is an unbounded amount of input data available to it, but the examples can only be seen one by one. At each step, a new object x_n becomes available from an unknown distribution and TCP computes a prediction set for this object. Then the true label y_n also becomes available, and the following error assessment is performed: if y_n is not in the prediction set

output by TCP, the prediction is marked as erroneous. An important observation is that once the true label for x_n is disclosed and the error assessment is done, the example z_n is added to the bag of previously seen examples and it is used for all future predictions done by the algorithm.

TCP has two very powerful properties and one undeniable drawback: first, it becomes more accurate as the execution of the algorithm progresses *i.e.* the number of examples collected grows and hence predictions are refined; second, the predictions done at each step i do not only consider the previously seen examples as most learning methods do, but also they are tailored to the object x_i. The main drawback of TCP is that it is computationally infeasible even for small input data. Thus, for practical applications, a compromise on the benefits offered by the transductive setting is necessary, for the sake of computational efficiency.

4.2.6 Inductive Computational Setting

Even though theoretically appealing, TCP cannot be used for modern datasets as the cost of computing non-conformity becomes overwhelming as the number of examples grows. A more forgiving computational setting is that of *Inductive Conformal Prediction* (ICP) ([106]) where the concept of induction replaces that of transduction. In the inductive model, a fixed number (or fraction) of examples is used for computing non-conformity, and this knowledge persists in the form of a learning model that is trained only once. After obtaining such model, we can measure non-conformity for all new examples we are presented with, without the necessity of performing expensive model training for each of them. Furthermore, with no extra work, we can use the inductive version of CP in an off-line (batch) setting, which is standard in machine learning.

The definitions of CP we gave in Section 4.2.3 were focused on the transductive setting, which is the most natural model for conformal prediction. If we want to switch to the inductive model, however, some formal modifications are required.

For defining ICP, let $Z^n := \wr z_1, z_2, ..., z_n \int$ be a *training bag* of examples that is fully available to us. We start by splitting Z^n into two non-overlapping bags: Z^k

and Z^l such that Z^k contains the first k examples in Z^n and Z^l contains the last l examples in Z^n [2], with $Z^k \cup Z^l = Z^n$ and $k + l = n$. We call Z^k the *proper training set* [3] and Z^l the *calibration set*. This split is fundamental to the efficiency of ICP because the training of the learning model that is used in the description of the non-conformity measure $\mathcal{A}$ will only be done once, over the fixed proper training set Z^k. Therefore, we write $\mathcal{A}_{Z^k}(z)$ as shorthand for $\mathcal{A}(Z^k, z)$ to represent the non-conformity of some example $z \in Z$ with respect to the examples in the proper training set *i.e.* $\mathcal{A}$ is parameterised by the proper training set Z^k.

Thus, for each $y \in Y$ and for an object $x \in X$, we re-define the computation of the non-conformity scores in (4.2) and (4.3) as:

$$\alpha_i = \mathcal{A}_{Z^k}(z_i) \;\; \forall z_i \in Z^l \tag{4.6}$$

and

$$\alpha = \mathcal{A}_{Z^k}((x, y)), \tag{4.7}$$

respectively. We also need to adjust the definition of the conformal algorithm, shown in (4.4), as follows:

$$p_y := \frac{|\{i = 1, 2, .., l : \alpha_i \geq \alpha\}| + 1}{l + 1} \tag{4.8}$$

ICP only computes the non-conformity scores for the examples in Z^l, the calibration set.

In short, ICP finds how the calibration set conforms to the proper training set; this is captured by the sequence of α_i in (Equation 4.6). Then, the algorithm finds how the new object x conforms to the proper training set when it is labelled with the postulated label y (Equation (4.7)). Finally, the conformal algorithm

[2] Notice that our bag notation is robust enough to support ordering/indexing of its elements, and we are making use of that feature. Thus, $Z^n := (z_1, z_2, ...)$ immutably references the first, second, etc, elements in the bag irrespective of the specific examples occupying those positions.

[3] Even though mathematically defined as a bag, we call it "set" for the sake of consistency with standard machine learning terminology.

produces a p-value for y by comparing the non-conformity of the tuple (x, y) against that of the examples in Z^l (Equation (4.8)).

Notice that, to avoid over-fitting, ICP never computes non-conformity scores for examples that were used for training the learning model encapsulated in $\mathcal{A}$ *i.e.* the model is trained over Z^k and the non-conformity scores α_i are computed only for those examples in Z^l.

By inspecting the modifications done to convert the TCP into ICP, we can see why the latter is much more computationally efficient. Notice that if we perform ICP in the online model, we might have to set some rules for re-computing the learning model once have seen "too many" examples in the input sequence. In this work, however, we assume that the inductive model for CP always runs in *off-line* fashion *i.e.* we are given the full *batch* of examples that are available for training the conformal predictor before any computation happens. This assumption is not restrictive as many single-point predictors in machine learning are designed to operate in this setting, logistic regression included. Algorithm 5 shows our protocol for ICP. When compared to the TCP protocol in Algorithm 4, we can see that now we can do a lot more computations without looking at our test examples yet. Also, because of the batch setting, we know exactly the number of predictions to be done by ICP, t in this case; and also we can see a more clear-cut separation between training and predicting.

Algorithm 5: Inductive Conformal Prediction Protocol.

Input: $\epsilon \in (0, 1)$: significance level, Z^n : input data, Z^t : test data

Define Z^k and Z^l *s.t.* $Z^k \cup Z^l = Z^n$ and $k + l = n$
Compute $\mathcal{A}_{Z^k}(z)\ \forall z \in Z^l$
$\mathcal{E}_0^\epsilon \leftarrow 0$
for $(x_i, y_i) \in Z^t$ **do**
    Compute prediction set $\Gamma_n^\epsilon \subseteq Y$
    if $y_i \in \Gamma_n^\epsilon$ **then**
        $err = 0$
    else
        $err = 1$
    end
    $\mathcal{E}_0^\epsilon = \mathcal{E}_0^\epsilon + err$
end

Finally, notice that ICP gives predictions that are not "tailored" to the new object we are attempting to classify. In fact, it only considers the examples in the proper training set. This puts ICP in disadvantage against TCP, which takes into consideration every example we have seen. In CP terminology, ICP has *weakened validity* and *degraded efficiency* compared to TCP ([12], Chapter 2, p. 27); however, this can be seen as a fair compromise if we are interested in using CP in more realistic scenarios.

Even though ICP is much more computationally efficient than TCP, it still faces considerable scalability issues. Then, the question remains: "can we do better?". The answer to this question is indeed affirmative, and we will discuss it on Section 4.3; specifically, we will present Coreset-based Conformal Prediction (CBCP), a new strategy that greatly enhances the running time of ICP while retaining learning performance. The most appealing aspect of our approach is that we do not need to write a new method from scratch; instead, we can use the theory of coresets to compress the input data approximately correctly, and keep CP untouched in the process. Before diving into our solution to make CP more scalable, however, we need to address two fundamental topics related to CP: the first one is regarding the definition of non-conformity measures for our classification problems of interest: LR and SVM; the second topic is about the definition of "learning performance" in the context of CP; that is, we will define performance metrics in order to evaluate the quality of the prediction sets produced by CP. These will be the topics for Section 4.2.7 and Section 4.2.8, respectively.

4.2.7 Non-conformity Measures for LR and SVM

In order to instantiate CP for the machine learning problems presented in Section 2.3, it is necessary to train a learning model on the proper training set. Such a model can be obtained by solving the corresponding optimisation problem associated with the ML problem of interest *e.g.* LR, SVM, etc. As explained earlier, CP uses existing machine learning techniques to measure conformity among the objects in the input data. The connection between single-point predictors and conformal predictors

then is encapsulated in the concept of the non-conformity measure $\mathcal{A}$. Thus, each definition for $\mathcal{A}$ results in different conformal predictors. For classification problems, a well-defined approach for defining $\mathcal{A}$ in a classifier-agnostic fashion is to compute the *Margin Error Function* (MEF), which can be defined, for any bag of n examples Z^n and any example $z := (x, y)$, as follows:

$$\mathcal{A}(Z^n, z) = 0.5 - \frac{\hat{P}(\hat{y}|x) - max_{y\,!=\hat{y}}\hat{P}(y|x)}{2} \tag{4.9}$$

where y is the true label from example z, $\hat{y}$ is an estimated label for the object x from example z, and $\hat{P}$ stands for an estimated probability function.

Equation 4.9 requires some unpacking: first, it should be useful to recall that the functional definition of the non-conformity measure is $\mathcal{A} : Z^* \times Z \to \mathbb{R}$; second, the equation requires to evaluate the probability estimates for object x. Therefore, to use MEF as a strategy for computing non-conformity, we need to define a computable probability function $\hat{P}$ for each of the learning problems in question. We first define this function for the problem of LR.

Probability Estimate Function for LR

Defining $\hat{P}$ for the problem of LR is relatively straightforward as the solution for the LR problem is in itself able to produce well-calibrated probabilities. Thus, for logistic regression, we define $\hat{P}_{LR}(y|x)$ as follows:

$$\begin{cases} 1\,/\,1 + e^{-\hat{\theta}\cdot x}, & \text{if } y = 1 \\ 1\,/\,1 + e^{\,\hat{\theta}\cdot x}, & \text{if } y = -1 \end{cases} \tag{4.10}$$

where $\hat{\theta}$ is the solution for the LR problem for the examples in Z^n as computed in Chapter 2, and $a \cdot b$ signifies the inner product between any two vectors a and b.

An interesting property of non-conformity measures is that they can be changed by any monotonic transformation without altering the resulting prediction sets. We can hence simplify the expression in (4.10) as:

$$\begin{cases} 1\,/\,-\hat{\theta}\cdot x, & \text{if } y = 1 \\ 1\,/\,\hat{\theta}\cdot x, & \text{if } y = -1 \end{cases} \tag{4.11}$$

Probability Estimate Function for SVM

Producing probability-type results with SVM is a less straightforward task as we will need to use a *calibrator*: a method for converting the output of non-probabilistic classifiers into well-calibrated probabilities.

In the RML family of algorithms, there is a powerful calibrator that produces *probability bounds* with strong theoretical guarantees with respect to the *true probability* of the predictions: *Venn-Abers Prediction* (VAP) ([14]). These calibration guarantee, however, can cost considerable amount of computing power; and since the purpose of this chapter is to accelerate the computation of CP, the VAP method is unfeasible at this point. The computational cost of VAP will be the main topic in Chapter 5, and there, this thesis will show that VAP can also benefit from data compression to get substantial computational speed-up. For now, however, this chapter will use a simpler calibrator to obtain $\hat{P}(y|x)$ for the SVM problem: Platt's scaling ([107]). After obtaining the probabilities, they will be used in Equation (4.9) to produce the non-conformity scores for each input point in the calibration set. Notice that since conformal prediction does not output probabilities, it does not require to have the strong calibration gurantees provided by a calibrator like VAP. Furthermore, $\hat{P}(y|x)$ is only necessary for computing the non-conformity scores α_i, and Equation (4.4) shows that all CP needs is to compare the values of the α_i against each other. Therefore, Platt's scaling should be sufficiently good for this particular task.

4.2.8 Performance Metrics

Now that conformal prediction has been fully defined, we present a set of criteria for measuring *how well* a conformal predictor performs. For the sake of exposition, given a bag with n training examples and the significance parameter ϵ, we will now consider Γ^ϵ to be a prediction set *i.e.* instead of $\Gamma^\epsilon(Z^n, x)$ as before, since the bag Z^n and the test object x will be clear from the context. Additionally, we consider we are given a sequence S containing t test objects $S := (x_1, x_2, ..., x_t)$, and that our goal is to calculate the average performance of the conformal predictor over this sequence.

As discussed previously, having guaranteed validity is not enough for the success of a conformal predictor. That is, we still need to make sure we get informative prediction sets, which is equivalent to asking for an efficient conformal predictor. Thus, the performance metrics used in CP measure efficiency from different perspectives. In this work, we are interested in the following criteria: the *S indicator*, the *N indicator*, the *singleton-rate indicator* and the *empty-rate indicator*. This set of measures has been extensively studied, and they have been successfully used to measure how informative the output of CP are (see [108] and [16], and references therein).

S Indicator

Proposed in [109], this measure is also referred as the *Sum criterion* as it gives the average sum of p-values p_y as follows:

$$\frac{1}{t}\sum_{i=1}^{t}\sum_{y\in Y} p_y^{(i)} \tag{4.12}$$

Notice that (i) in the above formula is simply an index to the test sequence S, not an exponent. The S indicator belong to a family of indicators called *ϵ-free measures*: metrics that do not depend on the significance parameter. Smaller values of this measure are preferable.

N Indicator

The N indicator ([110])([109]), or *Number criterion*, computes the average size of prediction sets over a test sequence:

$$\frac{1}{t}\sum_{i=1}^{t}|\Gamma_i^{\epsilon}| \tag{4.13}$$

In contrast with the S indicator, this is an *ϵ-dependent measure*. Smaller values of the N indicator are preferable.

Singleton-rate Indicator

This indicator computes the fraction of objects in the test sequence for which the conformal predictor outputs a prediction set Γ^ϵ that contains *exactly* one prediction. Formally:

$$\frac{1}{t}\sum_{i=1}^{t} \mathbb{1}_{\{|\Gamma^\epsilon|=1\}}, \tag{4.14}$$

where $\mathbb{1}_{\{\mathrm{P}\}}$ is an indicator function returning 1 if the predicate P is true and 0 otherwise. We will refer to this measure using the symbol $\mathcal{S}$. This ϵ-dependent measure achieves its best value when it equals 1 *i.e.* every single prediction set for every single test object contains exactly 1 prediction. Thus, the closer its value gets to 1, the better the performance of the conformal predictor.

Empty-rate Indicator

In similar fashion to the singleton-rate indicator, the empty-rate indicator accounts for the fraction of test objects for which the conformal predictor returns empty prediction sets. We can write it as:

$$\frac{1}{t}\sum_{i=1}^{t} \mathbb{1}_{\{\Gamma^\epsilon=\emptyset\}}, \tag{4.15}$$

where $\emptyset$ is the empty set. As the singleton-rate indicator, this is an ϵ-dependent measure; however, contrary to the singleton-rate indicator, we want this value to be as close to zero as possible.

4.3 Strategy: Compress the Proper Training Set

We have defined the framework of Conformal Prediction as a learning paradigm that we can use to obtain extra information regarding patterns we are interested in classifying [4]. Furthermore, CP can be seen as a statistical strategy for measuring the natural uncertainty that arises from making predictions. To generate this

[4]Notice that CP can also be used for the problem of regression, where the label space Y is the set of real numbers $\mathbb{R}$ (See Section 2.4.2); applying CP to regression, however, is out of the scope of this thesis.

extra information, CP demands non-negligible computing resources that quickly become exhausted for input data of moderate sizes. An important part of the computational burden can be eased if we switch to the inductive setting, at the cost of compromising the efficiency of the conformal predictors. In this section we go one step further; we present a new strategy to accelerate ICP by using the data-compression paradigm defined in Chapter 3: coresets. Thus, we will present a new learning protocol called *Coreset-based Conformal Prediction* (CBCP) that allows us to have conformal predictors that do not look for patterns in the big proper training set; instead, they learn over a very small, manageable, and accurate coreset, and by doing so, we will show that CBCP saves substantial computing time.

4.3.1 Coreset-based Inductive Conformal Prediction

Our main contribution in this chapter is to borrow the framework of coresets from the fields of Computational Geometry and Theoretical Computer Science, and put it into practice for accelerating ICP. Our line of work hence focuses on leaving predictors, both single-point and conformal, algorithmically intact, and concentrate on compressing the input data. As discussed in Chapter 3, there are more than one approach to obtain a coreset from the input data, which, in this case, is the proper training set for conformal predictors. We present in this section two state-of-the-art algorithms to construct coresets: the *Randomised Coreset Algorithm* (RCA) ([78]) for the LR problem and *Approximation Vector Machines* ([35]) for the SVM problem. The former is based on the sensitivity framework for constructing coresets and the latter follows the geometric decomposition approach for summarising data. Both RCA and AVM are defined and discussed in Chapter 3, and we will use them here as our main approaches for compressing the proper training set of ICP.

Let us then establish the notation for the method of Coreset-Based Inductive Conformal Prediction (C-ICP). C-ICP uses the same notation as CP; however, it also needs a pre-specified coreset algorithm that will be used for the compression of the proper training set. Thus, to distinguish prediction sets that have been computed from compressed data, we write $\Gamma^{\epsilon}_{\mathbb{S}}$ where $\mathbb{S} : Z^* \times \mathbb{N} \to Z^+$ is called

the *summary function*, Z^* is the space containing all bags of size n, $\mathbb{N}$ is the set of natural numbers and Z^+ is the space containing all bags of size m. The summary function takes a bag of size n and also a natural number m, the size of the desired summary. It then gives us a *compressed bag* of m examples and in order for the compression to be meaningful, of course, $m \ll n$ should hold. We should make two important observations regarding our new notation: (i) notice that CBCP allows for different coreset algorithms naturally; that is, we can use coreset algorithms in a plug-in-plug-out fashion, each coreset algorithm giving a different definition for $\mathbb{S}$; (ii) notice that the notation is robust enough for allowing any kind of compression technique to be used *i.e.* even compression functions that do not give a coreset guarantee. If $\mathbb{S}$ is indeed defined with a coreset algorithm, then this coreset construction should be for the machine learning problem *i.e.* for the single-point predictor, encapsulated in the non-conformity function $\mathcal{A}$.

To show C-ICP in action, we will define the summary function in two ways: one definition based on RCA algorithm and one based on the AVM algorithm.

4.3.2 C-ICP Protocol

The Coreset-based conformal prediction protocol in the inductive model (C-ICP) can be seen in Algorithm 6. Our approach takes advantage of the inductive model to add one extra layer of computation: the compression of the proper training set.

Notice that once we have constructed a coreset for the proper training set Z^k, we can proceed to discard it and from there on use only the coreset $\mathcal{C}$ for the remaining of the computations. Hence, we do not only save time in the non-conformity evaluation phase (line 3), but also we can potentially save space in memory.

In the next section we evaluate C-ICP on the data presented on Section 2.5, and we shall show that the strategy saves substantial computing time while retaining the predictive power of ICP.

Algorithm 6: Coreset-based Inductive Conformal Prediction (C-ICP) Protocol.

Input: $\epsilon \in (0,1)$: significance level, Z^n : input data, Z^t : test data

Define Z^k and Z^l *s.t.* $Z^k \cup Z^l = Z^n$ and $k + l = n$
Compute a coreset $\mathcal{C}$ for the proper training set Z^k
Compute $\mathcal{A}_{\mathcal{C}}(z)\ \forall z \in Z^l$
$\mathcal{E}_0^\epsilon \leftarrow 0$
for $(x_i, y_i) \in Z^t$ **do**
    Compute prediction set $\Gamma_{\mathbb{S}}^\epsilon \subseteq Y$
    if $y_i \in \Gamma_{\mathbb{S}}^\epsilon$ **then**
        $err = 0$
    else
        $err = 1$
    end
    $\mathcal{E}_0^\epsilon = \mathcal{E}_0^\epsilon + err$
end

4.4 Experiments and Results

Now that our main strategy have been formally defined, this section presents the results obtained from using our C-ICP protocol, presented in Algorithm 6. We test coreset-based inductive conformal predictors against standalone conformal predictors on the datasets presented in Table 2.1, Section 2.5.

Our experiments aim to shed light on the following research questions:

- **How do C-ICP perform computationally when compared to ICP?** We shall see that C-ICP does provide a substantial boost in terms of running time to standalone ICP. Furthermore, we will present cases where standalone ICP cannot be computed at all due to the size of some of our datasets. It is in those cases where C-ICP becomes the main strategy for obtaining the benefits offered by the CP framework.

- **Are validity and efficiency retained when the C-ICP protocol is applied?** This is a fundamental question; having fast approximation protocols is meaningless if predictive performance is arbitrarily lost. We will demonstrate that the efficiency of C-ICP, in the Conformal Prediction sense, does not exhibit any meaningful change when compared to ICP. Regarding validity, we

will see that the number of errors incurred by both methods are comparable, even though C-ICP measures non-conformity only over a small percentage of the proper training set.

4.4.1 Experiments Description

In order to test how the performance of C-ICP compares to that of ICP, we distinguish between the two methods as follows: **ICP**, naturally, represents the standalone ICP method that learns over the full proper training set. To represent C-ICP, we write C-ICP instances as **C-ICP**$_{g\%}$, meaning that, for this instance, the coreset size m is g percent of the proper training set size. Then, for example, **C-ICP**$_{1\%}$, **C-ICP**$_{3\%}$ and **C-ICP**$_{5\%}$ are three instances of the C-ICP protocol that use coresets of 1%, 3% and 5% the size of the proper training set, respectively.

Our evaluations compares the below two methods:

- **ICP:** refers to using standalone ICP over the full input data.
- **C-ICP**$_{g\%}$**:** refers to using the C-ICP protocol instantiated with a coreset of size g% of the proper training set.

We conduct our experiments as follows: for each dataset in Table 2.1, and for each of the three methods specified above, we perform the following computations 10 times:

1. **shuffle the data:** aiming at obtaining the best possible generalisation for our results, we do a full shuffling of the data before performing any computation.
2. **split the data:** we split the samples as described in Section 4.2.6, that is, we split it into the proper training set, which will be used to fit a single-point predictor to measure non-conformity, the calibration set, which will be used to calibrate the predictions, and the test set, which will be used to evaluate the predictions.

3. **measure non-conformity:** here is where standalone ICP differs from the C-ICP instances. For ICP we proceed as usual: we train a single-point predictor over the proper training set and use it to assign non-conformity scores to the calibration set examples. For C-ICP, however, we follow the corresponding protocol and compress the input data with a coreset algorithm: RCA (Algorithm 3) for LR and AVM for SVM. The small coreset is then used for computing the non-conformity scores.

4. **make predictions and evaluate performance:** the last step is to make predictions for the test examples and apply our performance metrics. We will use the criteria defined in Section 4.2.8.

Regarding the hardware, the experiments were deployed and run on a single desktop machine running the Linux operating system, equipped with an Intel(R) Xeon(R) CPU E3-1225 v5 @ 3.30GHz processor and 32 Gigabytes of RAM.

For the coreset implementations, we have used as baselines the implementations provided by Huggins *et al.* [5], for the RCA algorithm, and by Le *et al.* [6], for AVM. We have adapted these implementations to our needs for our experiments. For Conformal Prediction, we used the Nonconformist library [7]. All our programs were written using the Python programming language.

4.4.2 Evaluation Criteria

The evaluation of our results is done by considering the following criteria, which cover all of the computational aspects discussed in this chapter.

- **Computing time:** as we are deeply concerned about computational efficiency, our main criterion is the time, in seconds, that is needed for each of the considered approaches to compute prediction sets for each example 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. For the coreset-based predictors,

[5]https://bitbucket.org/jhhuggins/lrcoresets/src/master - last accessed 18/11/2021
[6]https://github.com/tund/avm - last accessed 18/11/2021
[7]https://github.com/donlnz/nonconformist - last accessed 18/11/2021

we also take into account the compression time, and all the computations it involves *e.g.* for RCA, the computation of a clustering of the data; for AVM, the computation of the enclosing balls. For the efficiency of the experiments, we have defined a time budget of 7,200 seconds for each of the considered methods. If the computing time of any method surpassed this budget, the method was stopped and its computing time, and all the other metrics, were reported as incomputable for the time budget.

- **Validity:** the most important criterion, from the conformal-prediction point of view, is the validity of the prediction sets obtained, as define in Section 4.2.4. We already know that the ICP protocol produces valid prediction sets; however, as this is the first time coresets and conformal prediction are combined together, the validity of C-ICP is not obvious. The validity is measured by counting the average number of prediction errors incurred by predictors, and making sure that this number does not arbitrarily surpasses the error rate specified by ϵ.

- **Efficiency:** the efficiency of conformal predictors tells us how informative the obtained prediction sets are; very efficient predictors are highly desirable (see our discussion in Section 4.2.4). To measure this, we use the four efficiency measures defined in Section 4.2.8; namely, the S indicator, the N indicator, the singleton-rate indicator and the empty-rate indicator.

4.4.3 Computing Time

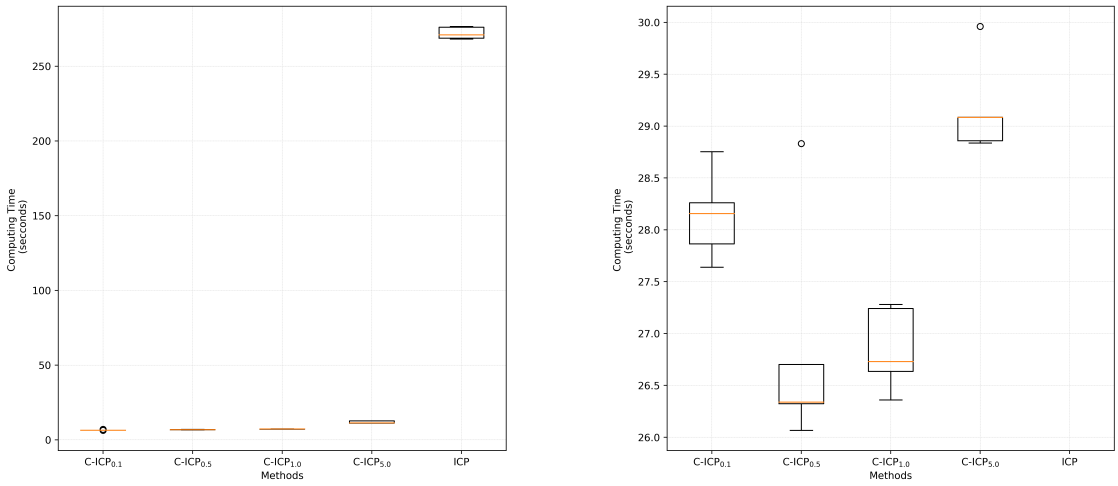
We start our analysis with the evaluation of the computational efficiency of C-ICP compared to that of ICP for the machine learning problems of logistic regression and support vector machines, both defined in Section 2.3.1 and Section 2.3.2, respectively. Notice that some of the coreset sizes used in C-ICP are very small; our intention is to verify if we can still maintain the same efficiency and validity of ICP when learning from a tightly compressed proper training set. It is also worth noting that we have used different coreset sizes for different data-sets; this is because, as data become

larger, we can allow for smaller coresets for C-ICP. On the other hand, if the data is not large, having a very small coreset could endanger the learning process; this is because very small coresets can miss all the examples of one of the two considered classes. Therefore, for covertype, higgs and webspam, C-ICP uses coreset sizes of 0.1, 0.5, 1 and 5 percent of the proper training set. For the smaller datasets, a9a, ijcnn1 and w8a, we use coreset of sizes 1, 5, 10 and 15 percent of the proper training set.

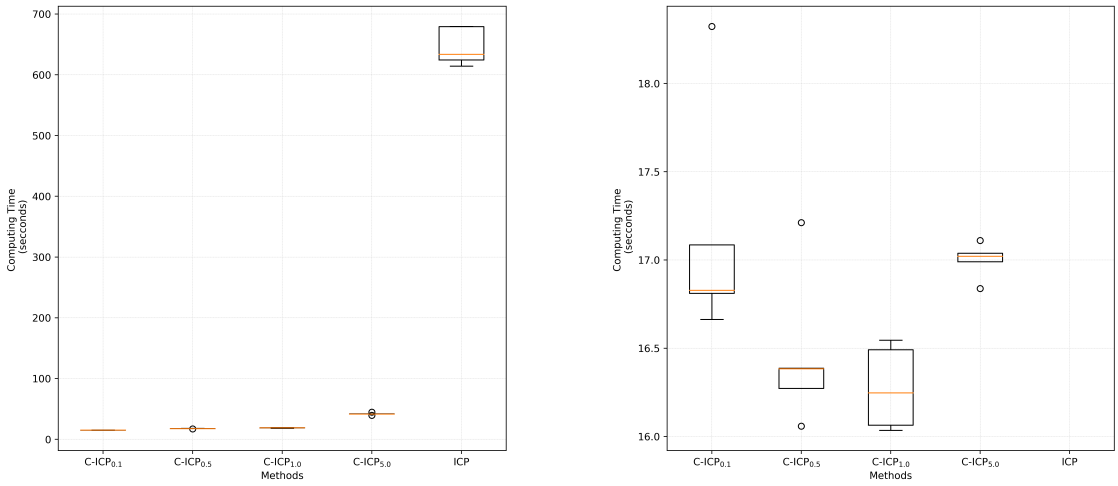
Table 4.1: Acceleration obtained by C-ICP for the six data-sets.

Data-set	Problem	Min acceleration rate	Max acceleration rate
a9a	LR	6.28x	12.78x
	SVM	110.42x	115.17x
covertype	LR	23.02x	41.54x
	SVM	-	
higgs	LR	-	
	SVM	-	
ijcnn1	LR	4.46x	7.38x
	SVM	99.79x	113.07x
w8a	LR	4.34x	13.91x
	SVM	2.68x	2.68x
webspam	LR	15.43x	42.83x
	SVM	-	

Figures 4.1, 4.2 and 4.3 show the computing time spent when running ICP and C-ICP for the six data-sets considered in this thesis. Each figure shows results for two data-sets, for both problems; LR, positioned on the left side, and SVM, positioned on the right side of the figures. For LR, C-ICP uses the RCA method (Algorithm 3, Section 3.5.2) to compress the proper training set into a coreset, while for SVM, AVM (Algorithm 10, Section 3.5.1) produces the desired summary of data within the C-ICP protocol. Each plot shows the time, in seconds, required to make predictions for the different instances on the x axis. Notice that the horizontal axis shows instances of C-ICP using different coreset sizes, and the standalone ICP method that learns over the full input data. The main purpose of presenting information in this fashion is to understand how sensible are the coreset



(a) covertype

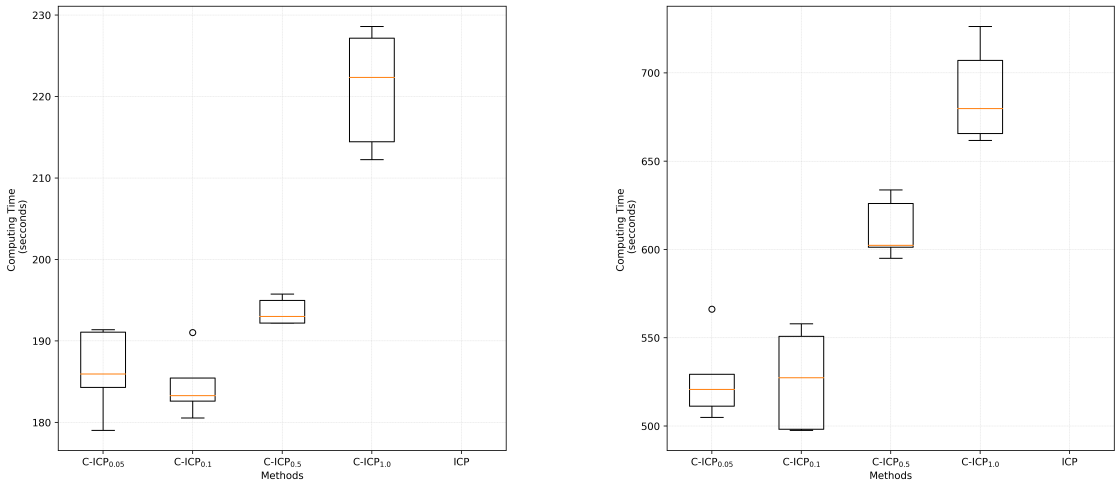


(b) webspam

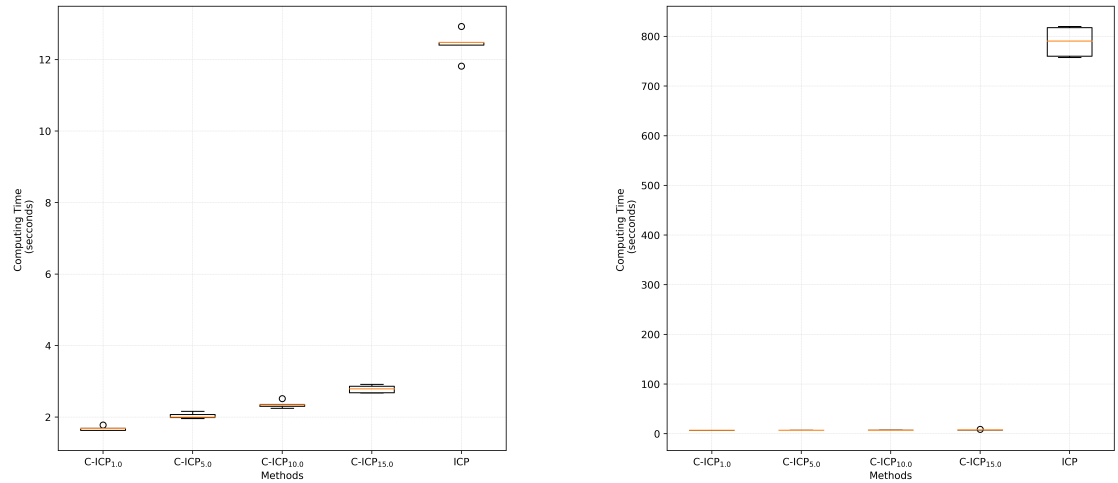
Figure 4.1: Comparison of the computing time between C-ICP and ICP on the covertype, webspam and higgs datasets. The RCA method is plugged in as a coreset algorithm. For the latter, VAP was incomputable within the time budget. The central line in each box indicates the median. The bottom and top of the box indicate the 25^{th} and 75^{th} percentiles, respectively. The small circles denote outliers. C-ICP does improve ICP's running time on all three datasets. Different coreset sizes does not seem to meaningfully impact on the computing time of C-ICP.

algorithms to the coreset sizes with respect to computing time, and how much of that computing time is saved compared to standalone ICP.

The first observation we need to make is that, aligned with our discussions in



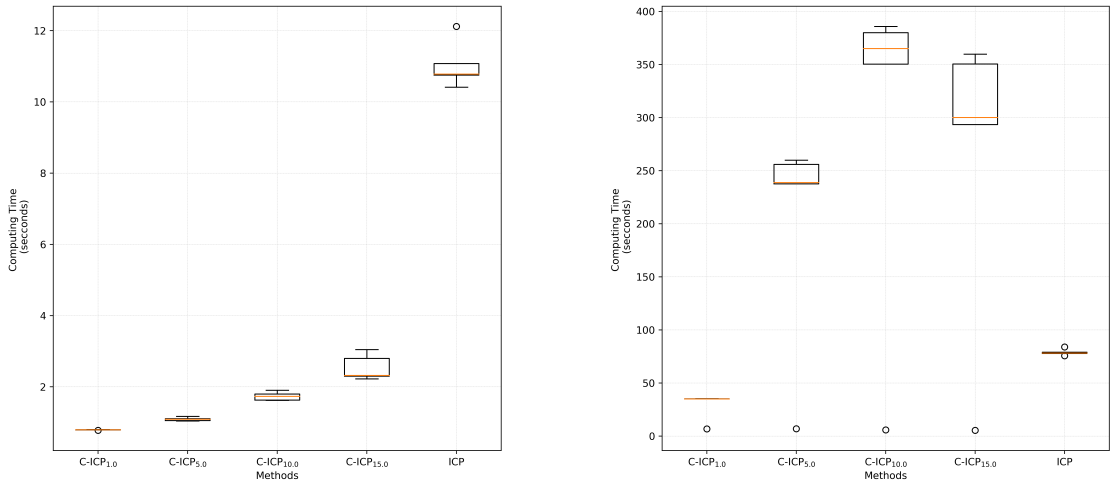
(a) higgs



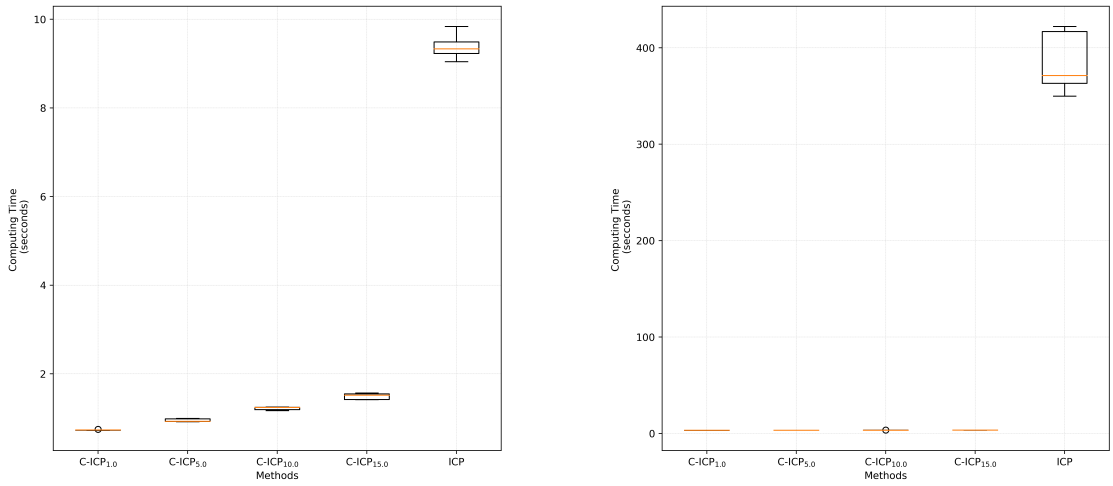
(b) ijcnn1

Figure 4.2: Comparison of the computing time between C-ICP and ICP on the covertype, webspam and higgs datasets. The RCA method is plugged in as a coreset algorithm. For the latter, VAP was incomputable within the time budget. 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 outliers. C-ICP does improve ICP's running time on all three datasets. Different coreset sizes does not seem to meaningfully impact on the computing time of C-ICP.

Chapter 2, the SVM problem is notably more computationally exhaustive than LR; we can see this by comparing the time scales on the y axes. Furthermore, for data-sets covertype, webspam and higgs, standalone ICP did not converge within



(a) w8a



(b) a9a

Figure 4.3: Comparison of the computing time between C-ICP and ICP on the covertype, webspam and higgs datasets. The RCA method is plugged in as a coreset algorithm. For the latter, VAP was incomputable within the time budget. 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 outliers. C-ICP does improve ICP's running time on all three datasets. Different coreset sizes does not seem to meaningfully impact on the computing time of C-ICP.

our time budget of 7,200 seconds for the SVM problem. This phenomenon only happened with the higgs data-set for LR.

Returning to our computing time analysis, we can indeed see that C-ICP boosts

the computing time of ICP by at least a factor of 2.69 (w8a) for SVM, and by a factor of 4.64 (ijcnn1) for LR. The maximum computable acceleration factor for the SVM case is 115.17 (a9a) and, for LR, it is 113.07 (ijcnn1). We write "computable" because, for the cases where no standalone ICP solution is available due to exceeding the time budget, the acceleration factors could not be exactly computed. For such cases, however, we can offer empirical lower bounds for the acceleration factor by using the time budget of 7,200 seconds. Thus, for svm, acceleration factors of *at least* 248.2, 400 and 10 correspond to covertype, webspam and higgs, respectively. The only incomputable case for LR is higgs, and it gives us an acceleration factor lower bound of 32. We remind the reader that these are lower bounds, and we cannot tell how loose or tight they are because we stopped the execution of ICP immediately after violating our time budget. Still, from our experimental observations, for these incomputable cases, we consider that the smaller values of lower bounds *e.g.* higgs, are loose ones; and those for covertype and webspam are tighter. This follows from the empirical fact that coresets seem to save more computing time as the input data grow larger. Table 4.1 presents exact minimum and maximum acceleration factors for the six data-sets, the incomputable cases are indicated with the - symbol.

Focusing on plot 4.3a from Figure 4.3, we can see that for w8a the computing time of C-ICP increases considerably with the coreset size, making C-ICP computationally more expensive than performing standalone ICP, for the SVM problem. This is an important phenomenon that we will extensively discuss in Chapter 6, and it generally happens when the machine learning task at hand can be efficiently solved for a specific data-set. There are two factors of the w8a data-set that combined could explain why the data compression process adds computational burden instead of providing acceleration: a) w8a is a small data-set b) w8a is almost completely sparse *i.e.* 96% of the input points are zero entries (See Table 2.1), and since sparseness makes learning an SVM classifier easier (see discussions on Section 2.3.2), then computing a coreset adds computational noise to the learning process. We can still see that acceleration is possible in this case *e.g.* when C-ICP uses a coreset of size 1% of the input data; but in general, small and sparse data seem to do better

without performing any data reduction. Complementary results on the computing time incurred by C-ICP can be found in Appendix A.

Our analysis so far indicate that C-ICP can meaningfully boost the performance of ICP for the presented data-sets. However, recalling the big data paradox (Section 2.2), favouring computational efficiency usually implies sacrificing performance. Therefore, computational acceleration is only half of our objective. In the next section we analyse the performance of inductive-conformal predictors when instantiated with and without compressed input data.

4.4.4 Efficiency and Validity

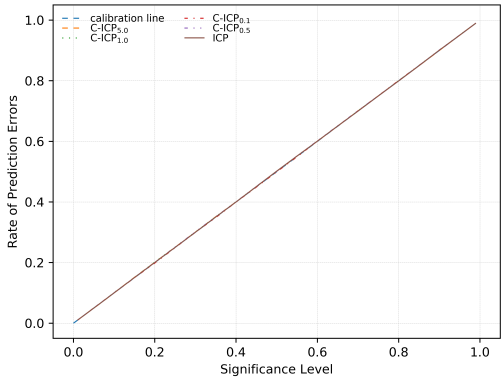
We now present the second part of our analysis: an examination on the validity and efficiency properties of C-ICP and how they compare to those obtained via ICP. Since, to the best of our knowledge, this is the first time an incursion on the analysis of performance of conformal predictors is done in the context of data compression, a clear-cut approach for making sense of the obtained result does not exist [8]. Thus, we conduct our analysis by considering the combination of all the efficiency measures defined in section 4.2.8 and the validity of the predictors; that is, even though each metric has its own importance, we believe that it makes more sense to assess performance using them as a system of metrics, and not individually.

From our CP definitions, it follows that every instance of CP is parameterised by a significance level ϵ. In order to make our analysis as complete as possible, we consider a wide range of significance levels. Namely, we assume that ϵ takes its values from the following increasing sequence:

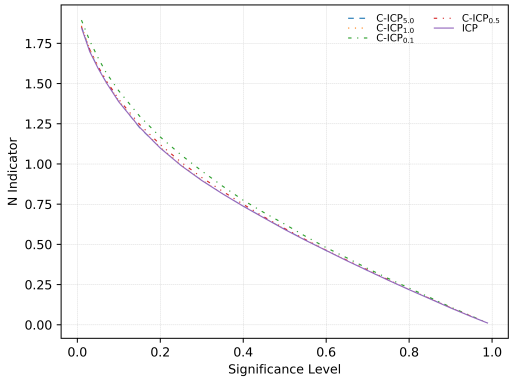
$$E = (0.01, 0.03, 0.05, 0.07, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45,$$
$$0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 0.99)$$

Therefore, for each $\epsilon \in E$, Figures 4.4, 4.5, 4.6, 4.7, 4.8 and 4.9 present the results of evaluating validity and efficiency metrics over the covertype, webspam, higgs, ijcnn1, a9a and w8a data-sets, respectively, assuming the LR problem. Notice

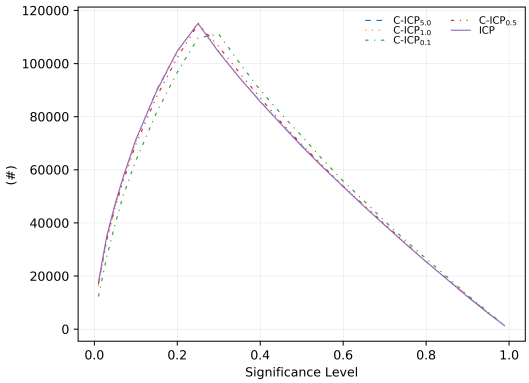
[8] technically, the first incursion was done in our previous works [31] and [111]. However, before these works, there is no evidence of previous incursions.



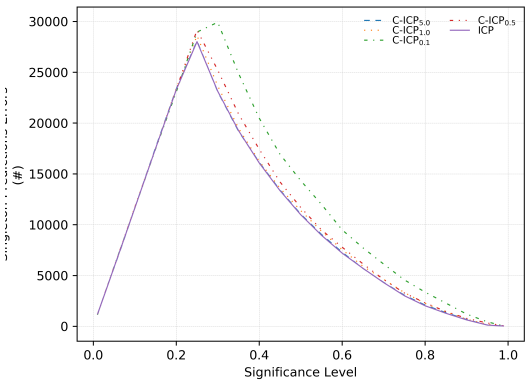
(a) Errors



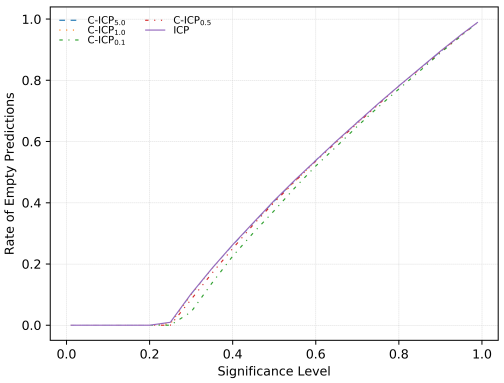
(b) Average Size of Prediction Sets (N Indicator)



(c) Singleton Predictions



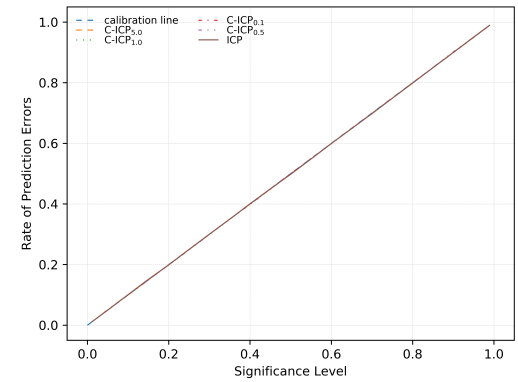
(d) Singleton Errors



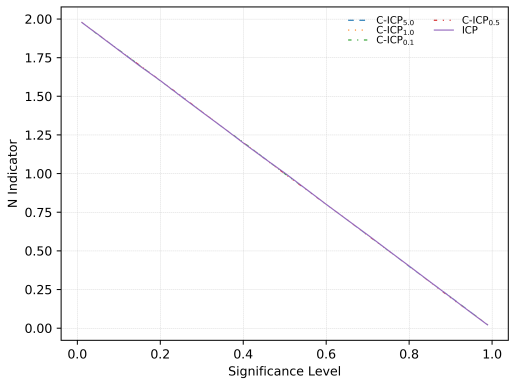
(e) Empty Predictions

Figure 4.4: Validity and efficiency measures obtained by C-ICP and ICP at different significance levels on the covertype data-set for the LR problem.

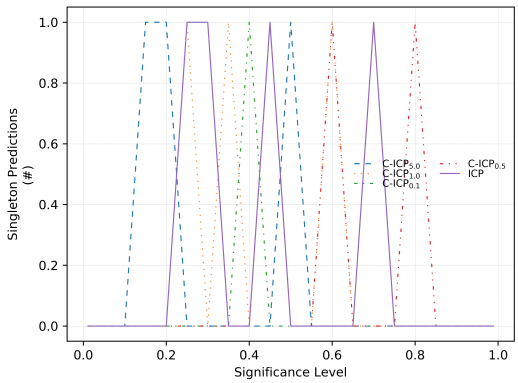
that since the patterns seem for LR and SVM are very similar, for the sake of space, we only include results for LR in this section. The figures containing the



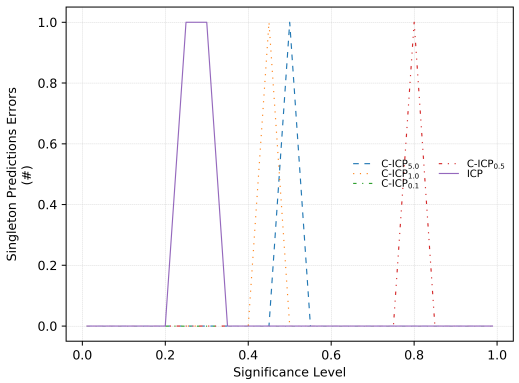
(a) Errors



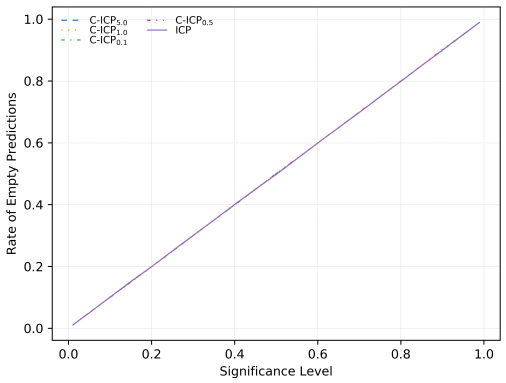
(b) Average Size of Prediction Sets (N Indicator)



(c) Singleton Predictions



(d) Singleton Errors



(e) Empty Predictions

Figure 4.5: Validity and efficiency measures obtained by C-ICP and ICP at different significance levels on the webspam data-set for the LR problem.

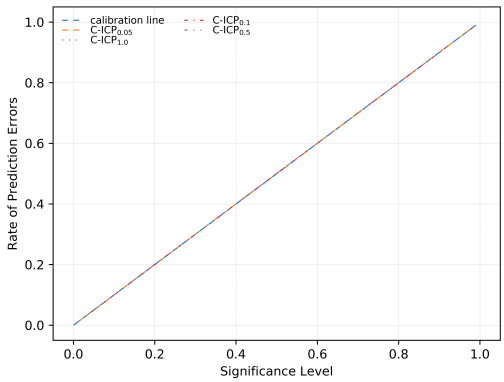
results for SVM can be found in Appendix A. However, the analysis presented here holds for both learning scenarios. Additionally, further results on C-ICP that

the reader may find useful can be found in the mentioned appendix *e.g.* using coresets as heuristics for both LR and SVM, highlights between uniform random sampling and coresets within CP, and more.

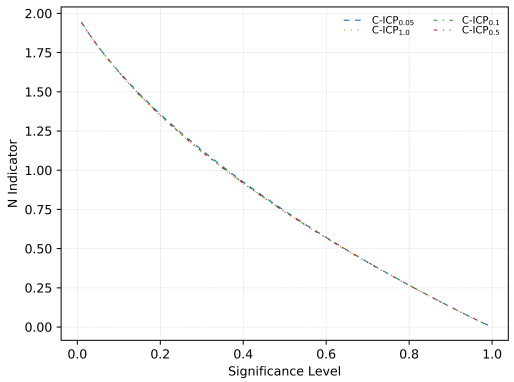
All plots in the figures have the significance levels on the x axis, and the corresponding metric values on the y axis. C-ICP was instantiated with different coreset sizes and, to remind the reader, that is indicated with the notation C-ICP$_{g\%}$ where g is the coreset size as a percentage of the proper training set. For the higgs data-set, we do not have a result for ICP as its computing time violated our time budget of 7,200 seconds. Thus, Figure 4.6 only shows results for C-ICP.

The first observation that can be extracted from the figures is that C-ICP retains exactly the same validity property when compared to standalone ICP, for all coreset sizes. This directly answers one of our fundamental research questions, stated in Chapter 1, asking whether CP can retain its validity property even when only a small summary of the proper training set is used. The validity results can be seen in plot **a** in the figures.

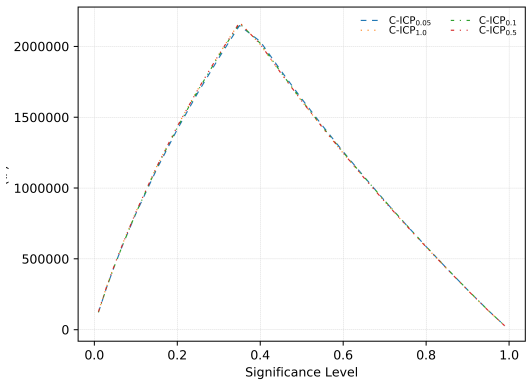
Next we turn our focus to the efficiency of the predictions made by C-ICP compared to those made by ICP. As we mentioned in Section 4.2.4, the informativeness of conformal predictors depends on how efficient the obtained prediction sets are. Our results indicate that for every data-set analysed, C-ICP retains the efficiency of standalone ICP. For example, we can see that in all data-sets, the N indicator values between the different instances of C-ICP and standalone ICP are virtually indistinguishable as all the curves have not only the same shape, but also the same values. We can also make the same observation for the number of empty predictions, for all the figures. Furthermore, we can appreciate how, as the significance level increases, the number of errors (plot **a** in figures) and the number of empty predictions (plot **e** in figures) both equally increase, while the N indicator, which shows the average size of prediction sets (plot **b** in figures) decreases. This corresponds with our CP definitions; that is, larger values for ϵ means that the CP can potentially *reject* a larger number of p-values, which in turn ends up giving a



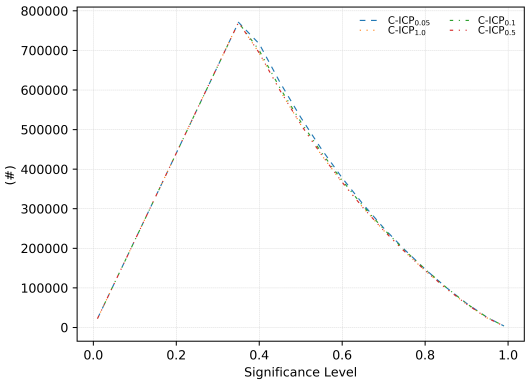
(a) Errors



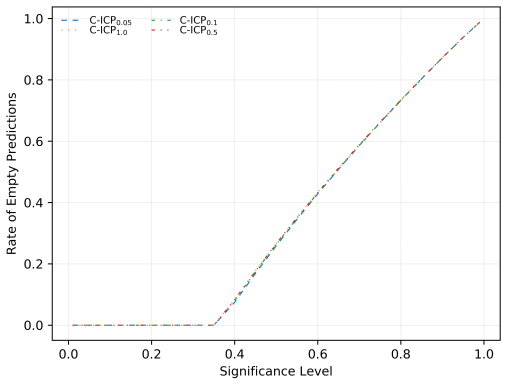
(b) Average Size of Prediction Sets (N Indicator)



(c) Singleton Predictions



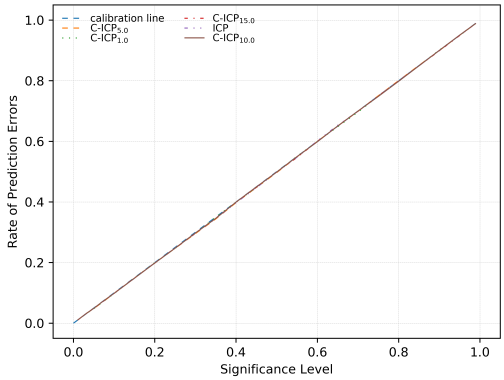
(d) Singleton Errors



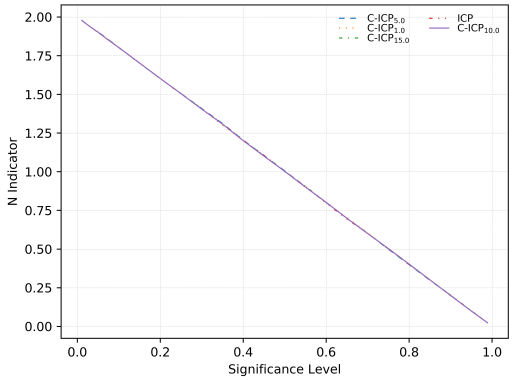
(e) Empty Predictions

Figure 4.6: Validity and efficiency measures obtained by C-ICP and ICP at different significance levels on the higgs data-set for the LR problem.

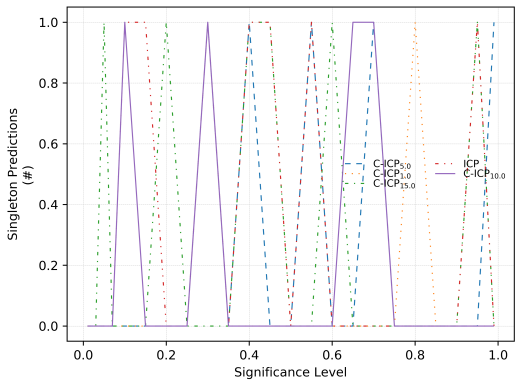
larger number of empty prediction sets. We can see that C-ICP behaves exactly as ICP with respect to these trends and relationships among metrics.



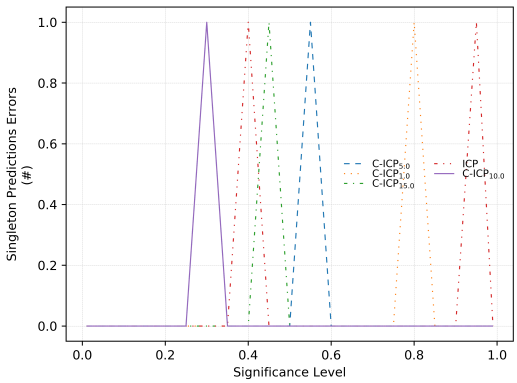
(a) Errors



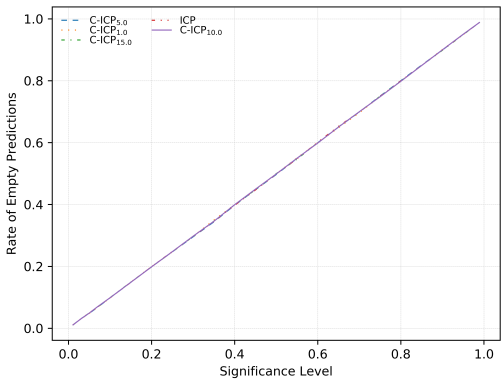
(b) Average Size of Prediction Sets (N Indicator)



(c) Singleton Predictions



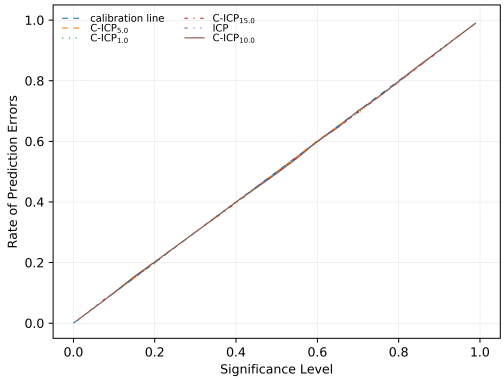
(d) Singleton Errors



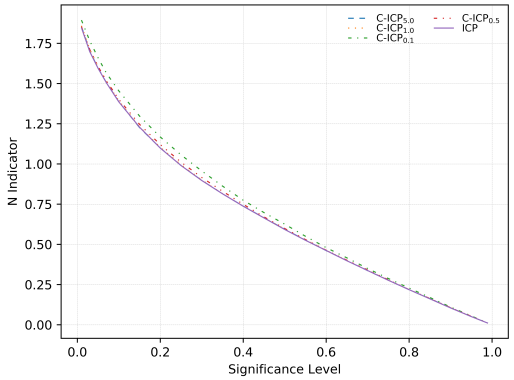
(e) Empty Predictions

Figure 4.7: Validity and efficiency measures obtained by C-ICP and ICP at different significance levels on the ijcnn1 data-set for the LR problem.

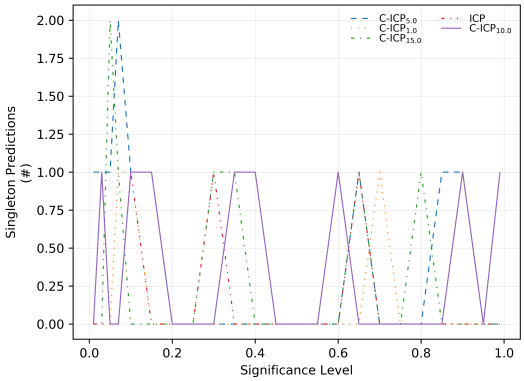
A very important efficiency indicator is the number of singleton predictions made by conformal predictors (plot **c** in figures), which we also complement with



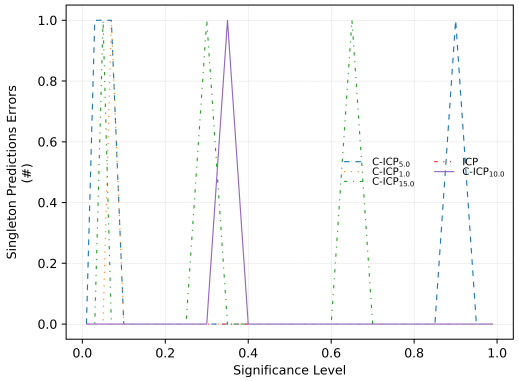
(a) Errors



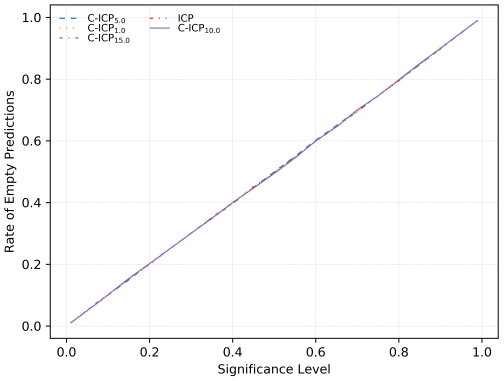
(b) Average Size of Prediction Sets (N Indicator)



(c) Singleton Predictions



(d) Singleton Errors



(e) Empty Predictions

Figure 4.8: Validity and efficiency measures obtained by C-ICP and ICP at different significance levels on the a9a data-set for the LR problem.

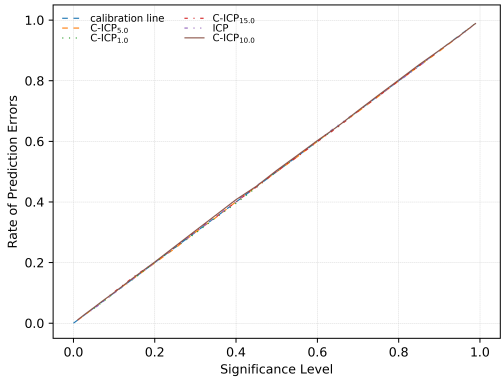
the number of cases where the single prediction made is incorrect (plot **d** in figures). For this pair of metrics, we have seen different behaviours in our data-sets.

Interestingly, for data-sets with high imbalance such as webspam (Figure 4.5), ijcnn1 (Figure 4.7), a9a (Figure 4.8) and w8a (Figure 4.9), both standalone ICP and C-ICP made a very small and irregular number of singleton predictions. Thus, the performances of the two methods cannot be distinguished from each other in those cases. This raises an issue that we believe is worth investigating as a future research problem: in supervised learning settings, existing coreset algorithms do not take into consideration the labelling of the input points. This seem to follow from the fact that coreset constructions were originally designed for unsupervised learning problems. However, we believe that using labelling information in the compression process could bring interesting gains in the performance of coresets. Based on these observations, we can formulate the following two research hypotheses that can inspire new research directions:

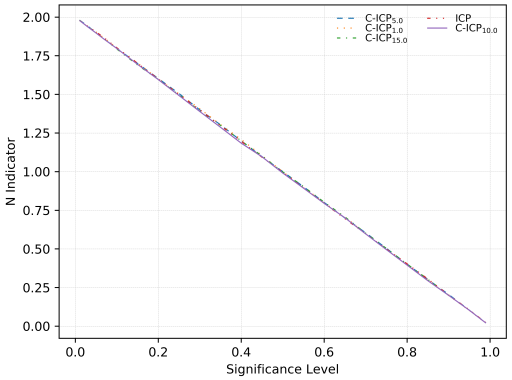
- coresets, in the supervised setting, are vulnerable to imbalance in data.
- conformal prediction needs a specialised approach for overcoming imbalance in the input data.

For the data-sets with a more balanced nature, covertype (Figure 4.4 and higgs (Figure 4.6), however, the behaviour of this tuple of metrics is more stable. For covertype, we indeed see that when C-ICP uses smaller coreset sizes, it does exhibit a slight loss in its capability of making singleton predictions. However, when C-ICP uses a coreset of size 5% of the proper training set, we can see that it achieves exactly the same performance as ICP, both in the number of singleton predictions and in the number of mistaken singleton predictions. For higgs, different coreset sizes in C-ICP do not seem to meaningfully affect the number of singleton predictions, although we can see that that the C-ICP instance using only 0.05% of the proper training set is slightly more prone to making singleton prediction mistakes.

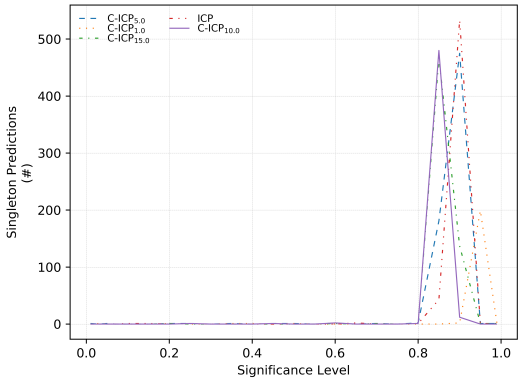
Finally, if we put together the results regarding computing time from the last section, and the results on validity and efficiency from this section, we can argue that the C-ICP protocol not only produces valid and efficient prediction sets, but also it gives its predictions very efficiently, in the computational sense. That is,



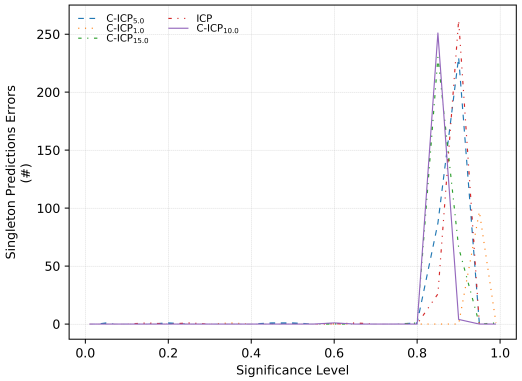
(a) Errors



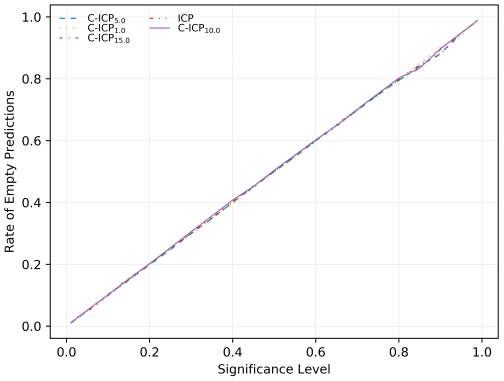
(b) Average Size of Prediction Sets (N Indicator)



(c) Singleton Predictions



(d) Singleton Errors



(e) Empty Predictions

Figure 4.9: Validity and efficiency measures obtained by C-ICP and ICP at different significance levels on the w8a data-set for the LR problem.

C-ICP can effectively retain the strong validity and the informativeness of ICP, while boosting the running time of inductive conformal prediction. Therefore, based

on our results, we can conclude that C-ICP is indeed the preferable approach when one is interested in using conformal prediction with data-sets of moderate sizes. Furthermore, our results suggest that as the input data becomes larger, the acceleration obtained via the compression process also increases.

4.5 Chapter Summary

Many machine learning algorithms are computationally expensive, making theit direct applications to large-scale datasets difficult or infeasible. The situation is exacerbated when we use those algorithms as part of a framework that attempts to enhance their output. Our experiments showed that obtaining the confidence levels for our predictions has to be paid with computing time that quickly becomes hopelessly draining for the hardware.

This chapter proposed Coreset-Based Conformal Prediction, a strategy for dealing with large-scale data within the conformal prediction framework. To the best of our knowledge, this is the very first attempt to use coresets along with CP. Coreset-based conformal predictors can use a small fraction of the original dataset, namely the coreset, and learn over it using conformal predictors. Our protocol leaves CP unmodified, and completely decouples the acceleration logic from the conformal predictors. We have observed that this was enough to accelerate computations by orders of magnitudes while avoiding losing meaningful predictive information. Furthermore, CBCP produces predictors that are valid, which shows that the CP theory still holds when we are dealing with compressed input data.

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; and that CBCP could potentially aid us in challenging scenarios where storage space is crucial.

The machines of nature, namely, living bodies, are still machines in their smallest parts ad infinitum.

— Gottfried Wilhelm Leibniz
Monadology and other philosophical writings ([112])

5 Inductive Venn-Abers Predictors with Enclosing Balls

5.1 Introduction

Chapter 4 has introduced the new learning protocol of Coreset-based Inductive Conformal Prediction (C-ICP), and it has shown that such protocol works well when one needs to measure uncertainty in classification problems using Conformal Prediction, and assuming input data of moderate size. In fact, for some data-sets, C-ICP was the only viable approach for using CP at all.

One fundamental step in CP is to define a non-conformity measure for the underlying machine learning predictor we are interested in using within the CP framework. Section 4.2.7 presented MEF (Margin Error Function) as a standard approach for defining non-conformity measures for single-point classifiers, such as LR and SVM. If one inspects the MEF definition, it becomes clear that *probability estimates* for points in the calibration set are needed in order to compute the respective non-conformity scores. Some single-point classifiers, like LR, output models that provide probabilistic interpretations for their predictions. Other predictors, like SVM, however, cannot be interpreted in terms of probabilistic estimates and hence a *calibrator* algorithm needs to be used as a mean to generate probability-type results. Venn-Abers Predictors (VAP) ([14]) is a framework within

the family of Reliable Machine Learning that can be used to produce well-calibrated probabilistic predictions using virtually any single-point classifier as a baseline. Furthermore, VAP does not only give us a probability estimate that, say, y is the correct label for object x; but it also reveals lower and upper bounds for the *ground truth* probability that y is the correct label for x.

As mentioned in Section 4.2.7, the computational cost of using VAP as calibrator can become exhaustive for input data of moderate size, which is the reason why Chapter 4.2 used a more rudimentary calibrator to obtain probability estimates for the non-conformity scores computations [1]. Thus, this issue raises the following research question: "Will VAP still produce well-calibrated probabilities if we instantiate it over a compressed version of its input data?"

This chapter is devoted to answer the above question when the problem of support vector machines is instantiated within the VAP framework. Loyal to the line of thinking presented in Chapter 4.2, the chapter advocates for the idea of focusing on reducing the volume of input data in order to achieve computational acceleration, instead of following the more traditional approach of focusing on the algorithms *themselves*. The result is a calibration protocol called *Venn-Abers Predictors with Enclosing Balls* (VAP-WEB), an approach that, analogous to what C-ICP does for CP, solves the computational struggle of the VAP framework by replacing the input data for SVM with a data summary that acts as a proxy to all the information we possess. To reduce the input data to a manageable size, we will apply once more the theory of coresets. Specifically, the AVM method, presented in Section 3.5.1, will be used to provably correctly approximate the input data for SVM by splitting the input space into a small number of *enclosing balls*.

The holy grail for the VAP-WEB protocol is to retain the high-quality calibration provided by standalone VAP, while at the same time saving as much computing time as possible.

[1] it is important for the reader to notice that we have indeed attempted to use VAP as calibrator for CP in the previous Chapter. This proved to be computationally infeasible, however. Hence the use of the simpler Platt's scaling method.

5.2 Background

The chapter starts with an overview of the theoretical background of IVAP. It may also be useful to revisit the main machine learning problem of this chapter: SVM; the details for this single-point predictor can be found in Section 2.3.2. Similarly, all the specifications regrading the enclosing-balls method for compressing data for SVM, implemented in the AVM (approximation vector machines) algorithm, can be consulted in Section 3.5.1.

5.2.1 Single-point Classifiers and Probabilities

It will be useful to start the presentation of Venn-Abers Predictors from the most atomic part of the framework: the *embedded* single-point classifiers. To refresh our discussions from previous chapters, we call single-point classifiers, or single-point predictors, to any machine learning method that can be used to find a solution to a classification problem. The advantage of this naming convention is that it emphasises the difference between standard predictors, that is, classifiers whose output is a single label $\hat{y}$ from the label space Y, and conformal predictors, classifiers whose output is *a set* of predictions, with each member of the set being a p-value offering information on the labels in Y that the conformal predictor cannot *reject*, given the information the predictor obtained from the training data.

VAP, just like CP, allows for virtually any single-point predictor to be plugged-in for the sake of *enriching* the outcome of the classifier. Let us focus on the underlying classifier. Following the definition of SVM in Chapter 2, we can see that the single-point predictor is, mathematically, a hyperplane [2] defined by $\theta \in \Theta$, and in the most basic classification scenario, we want to use it to classify a test object, call it x_{n+1}. To do so, we would need to compute a scalar $s_{n+1} = \theta \cdot x_{n+1}$ and then typically use some threshold to decide which label from Y will be chosen as the prediction $\hat{y}_{n+1}$. We have seen in Chapter 2 that, for example, SVM uses $sign(s_{n+1})$ to make its prediction, where $sign(a)$ outputs 1 if $a > 0$ and -1

[2] Remember that, as in the previous chapters, the bias term b is included in θ.

otherwise. This is the reason why single-point predictors are *scoring classifiers* ([14]), where the *score* is the scalar s_{n+1}.

Even though scoring classifiers are very useful, the information provided by them is somehow limited as they do not tell us much about the *probability* associated with each label in the label space.

In the next subsection, we describe the Inductive Venn-Abers (IVAP) method, a framework designed to complement scoring classifiers with well-calibrated probabilities on the labels; and as we shall see in Section 5.2.3, this extra information comes at a high computational cost.

5.2.2 Inductive Venn-Abers Predictors

The Venn-Abers method[3] is a machine learning framework designed to transform, or calibrate, the output of scoring classifiers into probabilities. It relies on the method of *Isotonic Regression* to calibrate the prediction scores obtained by a scoring classifier ([16]). For a more compact notation, let $\sigma : X \to \mathbb{R}$ be the *scoring function* that characterises a scoring classifier *i.e.* in our case, this function could be of the form $\sigma(x) := x \cdot \theta$, where θ is our kernel SVM solution. In order to transform the scores produced by $s(\cdot)$ into well-calibrated probabilities, the Venn-Abers procedure, in its inductive version, follows the below steps ([16]), which can be noticed to be reminiscent, to some extent, to those followed in ICP (Chapter 4.2.6):

1. Divide the input data into a *proper training set* of size k, and a *calibration set* of size l. Since our input size is n, we have $k + l := n$.

2. Train a scoring classifier on the proper training set and obtain a scoring function σ.

3. Find the scores $s_1, s_2, \ldots, s_l$ of the calibration objects $x_1, x_2, \ldots, x_l$.

4. For a test object x, compute its score s.

[3] "Venn" because the method is a form of *Venn Machine* ([13]), "Abers" after the surnames' initials of the authors in [113], which proposed the underlying technique used in the method.

5. Fit an isotonic regressor on $(s_1, y_1), (s_2, y_2), \ldots, (s_l, y_l), (s, -1)$ to obtain the function f_0. Fit another isotonic regressor on $(s_1, y_1), (s_2, y_2), \ldots, (s_l, y_l), (s, 1)$ to obtain the function f_1. The multi-probability prediction for observing the positive label $y = 1$ as the label for test object x is returned as the pair $(p_0, p_1) := (f_0(s), f_1(s))$.

Algorithm 7: The IVAP protocol.

Input: Z^n : input data, Z^t : test data
Output: Bag R with multi-probability predictions for examples in Z^t

begin
  Define Z^k and Z^l *s.t.* $Z^k \cup Z^l = Z^n$ and $k + l = n$
  $\sigma \leftarrow$ `trainClassifier(`Z^k`)`
  $S = \wr s_1, s_2, ..., s_l \wr \leftarrow$ `computeScores(`Z^l, σ`)`
  Define $I = \wr (s_1, y_1), (s_2, y_2), ..., (s_l, y_l) \wr$, where $s_i \in S$ and $y_i \in Z^l$
  $R = \emptyset$
  for $x \in Z^t$ **do**
    Compute the score for x as $s = \sigma(x)$
    Extend I with tuple $(s, -1)$ and call it I^- *i.e.* $I^- = I \cup \wr (s, -1) \wr$
    Extend I with tuple $(s, 1)$ and call it I^+ *i.e.* $I^+ = I \cup \wr (s, 1) \wr$
    Fit the isotonic regressors $f_0 =$ `trainIsotonoicRegressor(`I^-`)` and $f_1 =$ `trainIsotonoicRegressor(`I^+`)`
    $(p_0, p_1) \leftarrow (f_0(s), f_1(s))$
    $R = R \cup \wr (p_0, p_1) \wr$
  end
end

Notice that IVAPs give us multi-probabilistic predictions for each test object. We can interpret p_0 and p_1 as the lower and upper bounds, respectively, for the probability of predicting the positive label. Hence, ideally, we would like p_0 and p_1 to be close to each other. Algorithm 7 shows the formal protocol for the method. Also, IVAP is non-parametric when compared, for example, to conformal predictors; this is in the sense that IVAP does not expect any parameters as CP requires the significance level.

It is not obvious how to compare this kind of results against traditional point-probability results, or even how to apply standard probabilistic metrics to a multi-probabilistic output. We will see in Section 5.4.1 that there are well-defined procedures for turning the pair (p_0, p_1) into a point-probability p.

5.2.3 The Computational Cost

With the formal definitions for IVAP in place, we can now look into the central issue of this chapter: computational efficiency. Even though IVAP is the most computationally efficient version of VAP, we have indeed observed that scalability is a drawback when data-sets of moderate sizes are considered.

It is not hard to see that the computational efficiency of IVAPs depends heavily on the computational efficiency of the underlying scoring classifier. That is, an acceleration of the underlying classifier, SVM in this case, by definition implies acceleration for VAP.

Consider training our SVM classifier, for example, using the widely popular LIBSVM algorithm ([114]), which is the most famous implementation of Platt's *Sequential Minimal Optimization* (SMO) algorithm ([115]). Even though we are guaranteed to obtain a high quality solutions for support vector machines, and hence potentially good probability calibration via IVAP, the computational burden of LIBSVM will most likely overwhelm the IVAP framework for relatively large data-sets.

A reasonable alternative to LIBSVM is to attempt to train our underlying scoring classifier using the approach of *Stochastic Gradient Descent* (SGD) ([37]), which has seen substantial success in large-scale learning; however, even though the computations involve are quite fast, convergence can be very slow due to the noise introduced by its randomised nature ([51]).

In the next Section, we bring the already presented AVM algorithm, which is strongly based on SGD, but with the crucial difference that it learns over coresets.

5.3 Inductive Venn-Abers Predictors With Enclosing Balls

As anticipated, we address the computational bottleneck of the Venn-Abers framework by embracing the data-compression paradigm of coresets, which allows for the improvement of the computing time of algorithms, not by re-designing the

algorithms or even replacing them for faster ones, but by carefully reducing the input data size. Hence, the same potentially inefficient algorithm will surely terminate faster over reduced data. We call our method *Inductive Venn-Abers Prediction with Enclosing Balls* (IVAP-WEB); that is, IVAPs-WEB are IVAPs that learned the underlying SVM classifier over summarised data via the enclosing balls provided by the AVM algorithm, which was introduced in Chapter 3 and used on Chapter 4 as a *plug-in* coreset construction for the CBCP protocol. More specifically, the SVM model is learned over *core points*, each living in its own enclosing ball, as depicted in Figure 3.1 from Chapter 3.

In the IVAP-WEB protocol, as in C-ICP, it may seem counterproductive, and even dangerous, to restrict the underlying scoring classifier to *small data* given that we have *bigger data* as we know that in machine learning more data implies better predictive generalisation. In other words, we can see once more the *big data paradox*, discussed in Chapter 1.1, arising within the confines of VAP. However, this is the reason why we propose the small data summary of the big proper training set to be a coreset, and not just any arbitrary representation.With their strong *theoretical guarantees*, coresets will protect the underlying single-point predictor, a SVM model, from unpredictable decreases in predictive power. Coresets give us a systematic framework under which we can reduce our input data by keeping the most important data points with respect to an objective function f. In this Chapter, f is the *hinge loss* as defined on chapter 1.1.

5.3.1 IVAP-WEB Protocol

Algorithm 8 formalises the new IVAP-WEB protocol. Here, the scoring function for producing the classification scores is the *approximated* scoring function $\hat{\sigma}$, which was obtained via the enclosing balls approach implemented in AVM. Notice that the function that computes the approximated scoring function is parameterised by B; in SVM language, B is *the budget size* for the SVM model. In the context of IVAP-WEB, B is in fact the size of the coreset. By inspecting the details of the algorithm used to construct the coreset (Chapter 3), we see that B tells us the

number of *balls i.e.* hyper-spheres, that we are using to discretise the input space for the SVM problem. Furthermore, each of the B points in the coreset, the core points, is indeed the representative point for its particular ball.

Algorithm 8: The IVAP-WEB protocol.

Input: Z^n : input data, Z^t : test data
Output: Bag R with multi-probability predictions for examples in Z^t

begin
  Define Z^k and Z^l *s.t.* $Z^k \cup Z^l = Z^n$ and $k + l = n$
  $\hat{\sigma} \leftarrow$ `learnWithEnclosingBalls`$_B$`(`Z^k`)`
  $S = \wr s_1, s_2, ..., s_l \int \leftarrow$ `computeScores(`$Z^l, \hat{\sigma}$`)`
  Define $I = \wr (s_1, y_1), (s_2, y_2), ..., (s_l, y_l) \int$, where $s_i \in S$ and $y_i \in Z^l$
  $R = \emptyset$
  for $x \in Z^t$ **do**
    Compute the score for x as $s = \hat{\sigma}(x)$
    Extend I with tuple $(s, -1)$ and call it I^- *i.e.* $I^- = I \cup \wr (s, -1) \int$
    Extend I with tuple $(s, 1)$ and call it I^+ *i.e.* $I^+ = I \cup \wr (s, 1) \int$
    Fit the isotonic regressors $f_0 =$ `trainIsotonoicRegressor(`I^-`)` and $f_1 =$ `trainIsotonoicRegressor(`I^+`)`
    $(p_0, p_1) \leftarrow (f_0(s), f_1(s))$
    $R = R \cup \wr (p_0, p_1) \int$
  end
end

The main research questions of this chapter ask about the computing time saved by IVAP-WEB and whether the probability calibration obtained is comparable to that obtained from the full proper training set. In the next section, we provide our answers.

5.4 Experiments and Results

In this section we present the results obtained with IVAP-WEB and contrast it with those obtained using traditional IVAP, which deals with non-summarised data. As in the previous Chapter, we have tested our protocol over the data-sets presented on Chapter 2

Before jumping into the details on our experiments, let us refresh some important points. The purpose behind this set of experiments is to show that our protocol IVAP-WEB, described in Section 5.3, improves the running time of inductive Venn-Abers predictors when SVM is used as the underlying scoring classifier. Hence, our

experiments compare two approaches: *IVAP*, which considers inductive Venn-Abers prediction in the traditional sense as described in Section 5.2.2 *i.e.* we use the whole proper training set to train a SVM classifier; and *IVAP-WEB*, which is our proposed protocol to speed up IVAP.

Notice that each of the two approaches above needs an *optimisation solver* to train the SVM classifier. In the IVAP case, we used the previously mentioned LIBSVM solver ([114]) which has seen tremendous success in efficiently solving the SVM objective function, and indeed it is the state-of-the-art algorithm for computing a high quality SVM model for large data-sets. For IVAP-WEB, we obtain an approximately correct SVM solution using the AVM algorithm, which allows us to use enclosing balls to summarise the input data to find a SVM solution. Finally, for the sake of completeness, we include computations of obtaining a SVM model from an uniform random sample (URS) of the input points; the idea is to compare the coreset-based approach against the simpler URS method. Theoretically, coresets should perform better than URS, and in the worst case, these methods should perform equally.

5.4.1 Metrics

Our notion of performance is captured by the below three measures. It is useful to remember at this point that IVAP produces multi-probability outputs *i.e.* (p_0, p_1), and as anticipated in Section 5.2.2, we need to adapt these outputs to point-probability type of results. We followed the *minimax* approaches suggested by Vovk *et al.* ([14]). In particular, the authors give two strategies for combining (p_0, p_1) into p depending on the metric to be used [4]; for measuring MLE, we can obtain point-probability predictions by using $p := \frac{p_1}{1-p_0+p_1}$; and for Brier score, the minimax probability is $p := \bar{p} + (p_1 - p_0)\,(\frac{1}{2} - \bar{p})$, where $\bar{p} := (p_0 + p_1)/2$. Finally, notice that for all the metrics considered, small values are preferred over large ones.

[4]The MLE and Brier score formulations assume that labels are either 0 or 1; of course, we adapted this for our case *i.e.* $y_i \in \{1, -1\}, \forall i$.

- **Computing Time:** this is the number of seconds elapsed from the starting of the method till the end of its execution.

- **Mean Log Error (MLE):** this is the average of the per-point log loss suffered by some probabilistic predictor over a test set. Mathematically, the per-point log loss is defined as $L_{log}(y_i, p_i) := -(y_i \ln(p_i) + (1 - y_i)\ln(1 - p_i))$, where y_i is the true label for a test object x_i, and p_i is the predicted probability of the true label.

- **Brier Score:** this is the average of the per-point squared loss suffered by a probabilistic predictor over a test set. Formally, the per-point squared loss is defined as: $(y_i - p_i)^2$, where y_i is the true label for a test object x_i, and p_i is the predicted probability of the true label.

SVM Parameters

We designed our experiments considering that our underlying scoring classifier, SVM, is very sensitive to its input parameters. Hence, for the sake of fairness, we decided on the parameters' value as follows: for each method to be tested, and for each data-set, we performed a cross-validation phase over the proper training set to set all the involved parameters. The final parameters used, after doing a 5-fold cross-validation, were the best ones from the following values as in [35]: $\lambda \in \{\frac{2^{-4}}{n}, \frac{2^{-3}}{n}, ..., \frac{2^{16}}{n}\}$ and $\delta \in \{1.0, 2.0, ..., 10.0\}$ for AVM [5], where n is the size of the proper training set; $C \in \{2^{-2}, 2^{-1}, ..., 2^{10}\}$ for LIBSVM, and $\gamma \in \{2^{-8}, 2^{-7}, ..., 2^{8}\}$ for both AVM and LIBSVM *i.e.* both need the kernel coefficient. Table 5.1 gives a summary of the values used for each parameter of each method. Interested readers can refer to Appendix C for information regarding how the AVM parameters affect the final calibration obtained when using the IVAP-WEB protocol.

[5] LIBSVM always sets λ to 1/2.

Table 5.1: Parameters for IVAP and IVAP-WEB. IVAP uses the well-known LIBSVM solver while IVAP-WEB apply the coreset-based solver AVM.

Parameter	Algorithm(s)	Description
λ	LIBSVM | AVM	L_2 Regulariser.
γ	LIBSVM | AVM	RBF Kernel coefficient.
δ	AVM	Diameter of balls in input space.
B	AVM	Budget size for SVM model.
C	LIBSVM	Dual SVM Regulariser.

5.4.2 The Evaluation

The evaluation of IVAP and IVAP-WEB were done following the below steps. As mentioned previously, we also include a third approach which consists in applying IVAP over a small Uniform Random Sample (URS) of the proper training set. We call this method *IVAP-URS* and the size of the small sample is set to the same size as B in AVM.

- **Data splitting:** we leave 20% of the total number of data points for the test set. Then, we proceed to further divide the remaining 80% of points into 20% for calibration and 80% for proper training set.
- **Cross-validation:** we then perform 5-fold cross-validation for each of the three approaches. Due to computing limitations, we cross-validate using 1% of the proper training set. The parameter values were taken from the ranges discussed above, in Section 5.4.1.
- **Methods:** we run IVAP, IVAP-WEB and IVAP-URS with the best parameters found.
- **Metrics:** the metrics detailed in Section 5.4.1 are applied and their output are stored.

We repeat the above experiment 10 times and report averaged values for each of the metrics. For the sake of time, we have imposed a time budget for the IVAP and IVAP-WEB methods; we allowed a total of 7200 seconds for each

of the protocol to converge to a solution. If this threshold was exceeded, the computation of that algorithm was stopped and we reported that the time necessary for convergence was unbounded.

Regarding the implementations, all the methods and experiments were coded using the Python programming language. For LIBSVM and SGD, we used the well-known Scikit-Learn library https://scikit-learn.org/stable/. For computing IVAP, Toccaceli's implementation https://github.com/ptocca/VennABERS was used. Finally, our experiments were performed on a single desktop PC running Ubuntu Linux, equipped with an Intel(R) Xeon(R) 3.30GHz processor and 32 Gigabytes of RAM.

5.4.3 Results

Table 5.2: Performance comparison using the covertype data-set. δ and B are set to 5 and 1% for IVAP-WEB, respectively. IVAP-RUS uses a randomly chosen 10% of the proper training set.

Measure	IVAP	IVAP-WEB	IVAP-URS
Mean Log Error	-	**0.5785**	0.6795
Brier Score	-	**0.1965**	0.2433
Computing Time (s)	-	133.9570	**43.0821**

Probability Loss and Computing Time

In this section, we present and discuss the results obtained on probability calibration and computing time. Tables 5.2, 5.3, 5.4, 5.5, 5.7 and 5.6 show the performance of IVAP, IVAP-WEB and IVAP-URS for the six data-sets presented in Table 2.1. The most preferable results, for each measure, are highlighted in bold fonts.

First, it is important to mention that the numbers in the tables are the medians computed over 10 runs of the methods; thus, they are resistant to outliers or variances that could mislead the analysis. Also, notice that IVAP-URS always uses a sample size that matches the maximum number of core points allowed in IVAP-WEB *i.e.* B. This allows this method to be the fastest one in all cases; however,

since it trains SVM on a small arbitrary random sample of size B, the points chosen do not necessarily represent the proper training set adequately. This clearly affects the probability-type results of the Venn-Abers framework as it can be seen that IVAP-URS always obtains the worst MLE and Brier measures. IVAP-WEB, on the other hand, is not as fast as IVAP-URS, but its performance is in all cases preferable. Interestingly, despite using a very small fraction of the proper training set for learning a SVM classifier, IVAP-WEB incurs in smaller losses compared to IVAP for two out of the three data-sets for which VAP did not violate our time budget of 7,200 seconds. Namely, for a9a the results show that IVAP-WEB has a better performance in both MLE and Brier metrics; and for ijcnn1, we see that IVAP-WEB performs better with respect to the Brier score, although the difference is small. This phenomenon is aligned with some of our observation in the previous chapter *i.e.* it seems that coresets can sometimes help in removing *noise* from the data, leaving only the most important information for the learning problem in question. We believe this issue should be the baseline for future investigations on the empirical performance of coresets.

Table 5.3: Performance comparison using the higgs data-set. δ and B are set to 5 and 1% for IVAP-WEB, respectively. IVAP-RUS uses a randomly chosen 10% of the proper training set.

Measure	**IVAP**	**IVAP-WEB**	**IVAP-URS**
Mean Log Error	-	**0.6864**	0.6913
Brier Score	-	**0.2467**	0.2491
Computing Time (s)	-	1146.58	**266.39**

For w8a, LIBSVM obtains the best quality solution by a large margin. Even in this case, it might be very useful in some applications to consider IVAP-WEB for a faster solution with a compromise in quality.

For the larger data-sets in Tables 5.2, 5.2 and 5.2, we can only compare IVAP-WEB against IVAP-URS as standalone IVAP was too expensive to converge within our predefined time budget. As we mentioned earlier, in all cases IVAP-WEB performs better than IVAP-URS; however, in the case of higgs, the difference

between the two methods is very small. This fact can reveal some interesting aspects regarding this particular data-set: it seams that its data points are close to a uniform distribution with respect to SVM objective. Thus, in this case, compressing the input data seems to almost degenerate into taking a uniform random sample. The situation is very different for webspam, where IVAP-WEB substantially improves the results of the simple URS approach.

Table 5.4: Performance comparison using the webspam data-set. δ and B are set to 5 and 1% for IVAP-WEB, respectively. IVAP-RUS uses a randomly chosen 10% of the proper training set.

Measure	**IVAP**	**IVAP-WEB**	**IVAP-URS**
Mean Log Error	-	**0.1177**	0.6516
Brier Score	-	**0.0343**	0.2298
Computing Time (s)	-	940.4043	**22.6922**

Table 5.5: Performance comparison using the a9a data-set. δ and B are set to 9 and 10% for IVAP-WEB, respectively. IVAP-RUS uses a randomly chosen 10% of the proper training set.

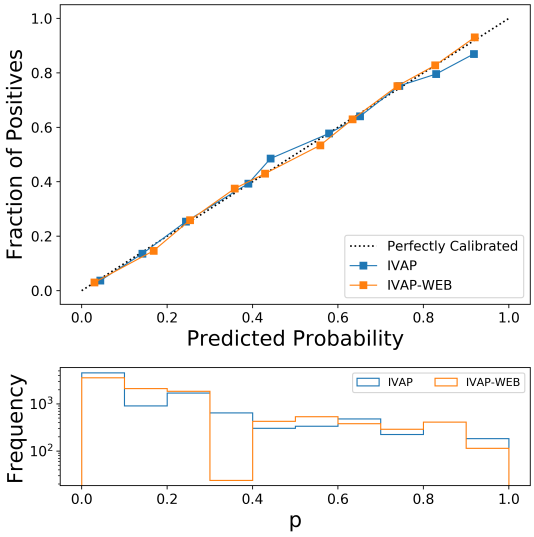
Metric	**IVAP**	**IVAP-WEB**	**IVAP-URS**
Mean Log Error	0.373	**0.351**	0.547
Brier Score	0.117	**0.112**	0.180
Computing Time (s)	586.6	37.0	**18.0**

Table 5.6: Performance comparison using the w8a data-set. δ and B are set to 21 and 10% for IVAP-WEB, respectively. IVAP-RUS uses a randomly chosen 10% of the proper training set.

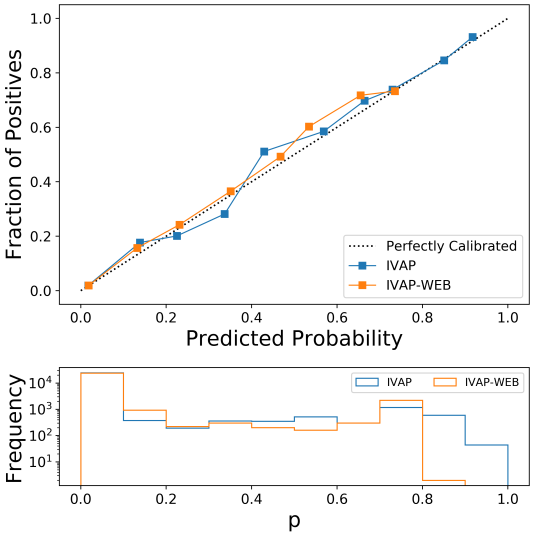
Metric	**IVAP**	**IVAP-WEB**	**IVAP-URS**
Mean Log Error	**0.0465**	0.0710	0.135
Brier Score	**0.0100**	0.0136	0.0291
Computing Time (s)	226.6	136.3	**78.6**

Table 5.7: Performance comparison using the ijcnn1 data-set. δ and B are set to 9 and 10% for IVAP-WEB, respectively. IVAP-RUS uses a randomly chosen 10% of the proper training set.

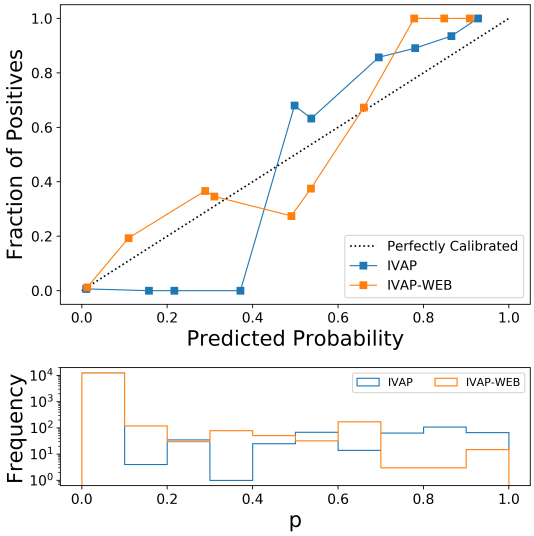
Measure	IVAP	IVAP-WEB	IVAP-URS
Mean Log Error	**0.149**	0.166	0.547
Brier Score	0.117	**0.112**	0.180
Computing Time (s)	585.4	16.21	**18.0**



(a) a9a data-set.



(b) ijcnn1 data-set.

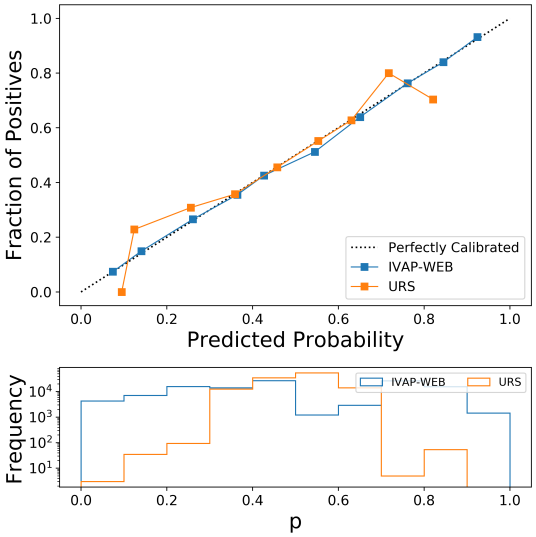


(c) w8a data-set.

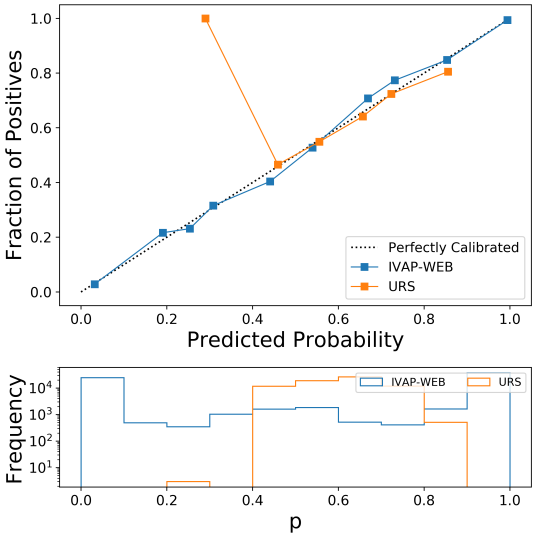
Figure 5.1: Calibration plots for a9a, w8a and ijcnn1. They demonstrate that IVAP-WEB predictions are well-calibrated.

Calibration Curves

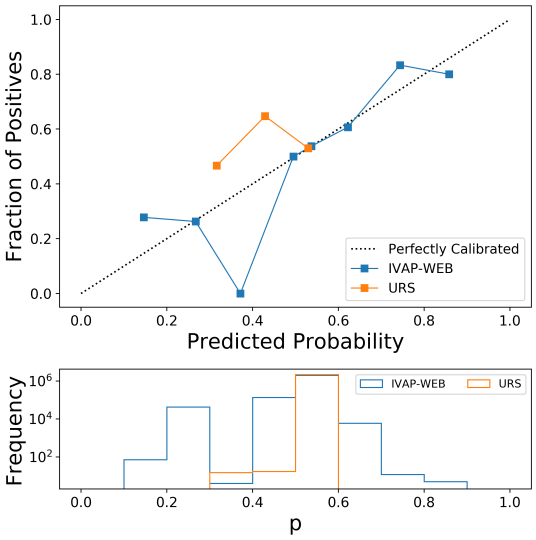
We then evaluate the calibration of IVAP and IVAP-WEB, which can be seen in Figures 5.1 and 5.2. Notice that the former groups the smaller data-sets while the latter contains the bigger ones. As a result, Figure 5.2 does not show information regarding IVAP as the time necessary for computing it surpassed the time budget defined for the experiments.



(a) covertype data-set.



(b) webspam data-set.



(c) higgs data-set.

Figure 5.2: Calibration plots for covertype, webspan and higgs. They demonstrate that IVAP-WEB predictions are well-calibrated.

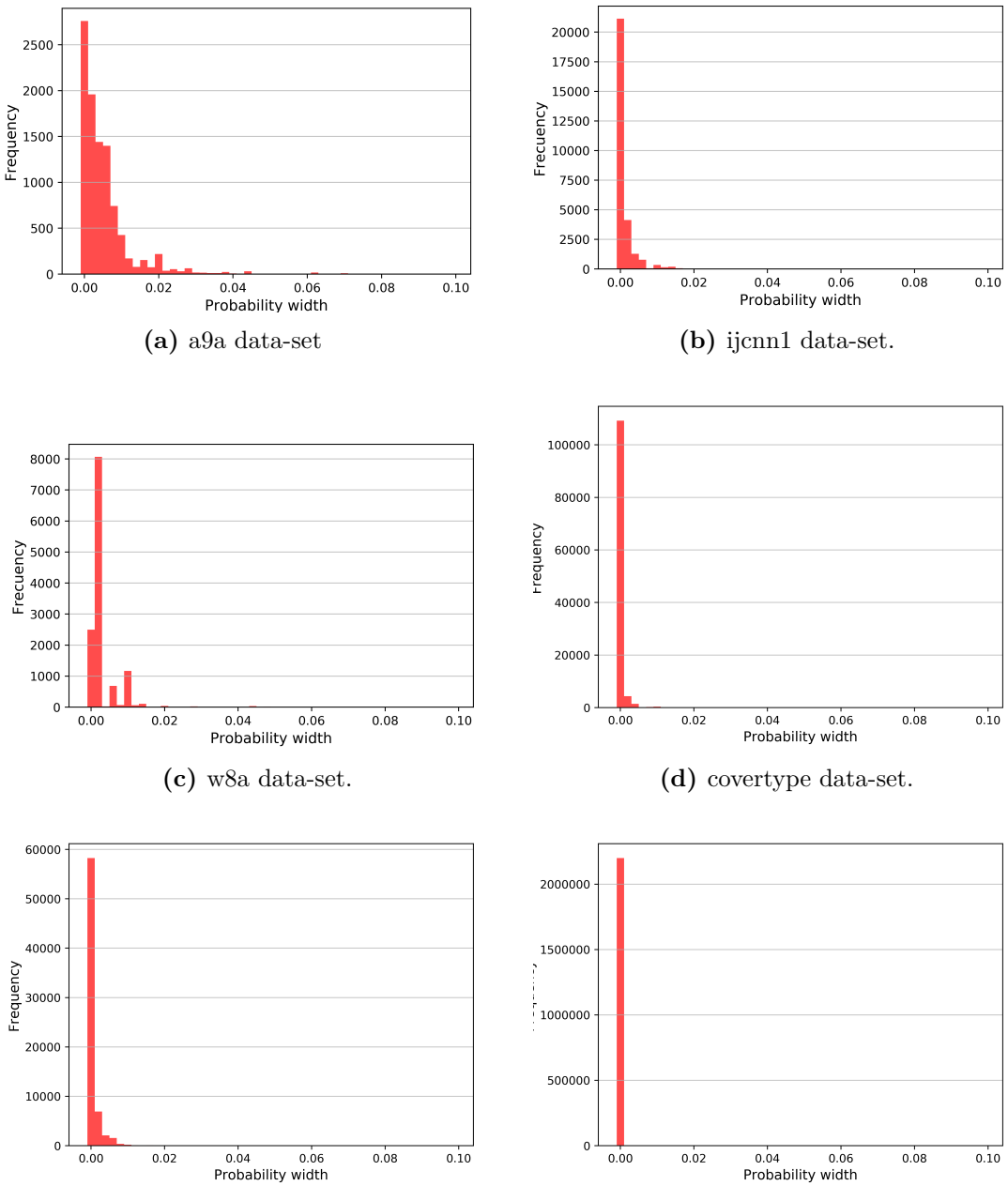
As mentioned in a previous section, Venn-Abers outputs an upper and lower bound, namely p_0 and p_1, for the probability of the positive class. We then need to combine these bounds for the analysis, obtaining a single value p. A calibration graph tells us how the predicted probability for the positive class matches the actual fraction of positives observed in the test set. The closer the curve gets to the perfect diagonal line, the better its calibration is. We can then appreciate that even though IVAP-WEB observes far less training points than IVAP, it still provides very well-calibrated probabilities. We can also see that w8a is a particularly difficult data-set for both methods as the two of them obtain poor calibration. We speculate that this has to do with the fact that w8a is a very sparse data-set *i.e.* 96% of its entries are zero. We further complement the calibration plots with histograms that show the distribution of p for our test set. Ideally, one would want the values of p to be concentrated near zero or one as this means the predictions are more certain.

Multi-probability Efficiency

Finally, the efficiency of both IVAP and IVAP-WEB are shown in Figures 5.3 and 5.4, respectively. In this context, the notion of efficiency is defined as the difference between p_0 and p_1; that is, the closer they are from each other, the more certain our predictor is and the more efficient the prediction is. We call the *width* of the prediction to the difference $|p_1 - p_0|$. In the figures, we can see that both methods achieve good efficiency and thus we can conclude that IVAP-WEB also largely retains the efficiency of IVAP.

5.5 Chapter Summary

This chapter discussed the usefulness of probability calibrators and presented the VAP framework, a systematic method for using a single-point classifier to produce well-calibrated probability-type results. We showed that the cost of running VAP follows directly from the computational efficiency of its underlying predictor. The SVM classifier is a fundamentally important method for classification in machine learning as its rule of finding large-margin data separators, combined with its



(a) a9a data-set

(b) ijcnn1 data-set.

(c) w8a data-set.

(d) covertype data-set.

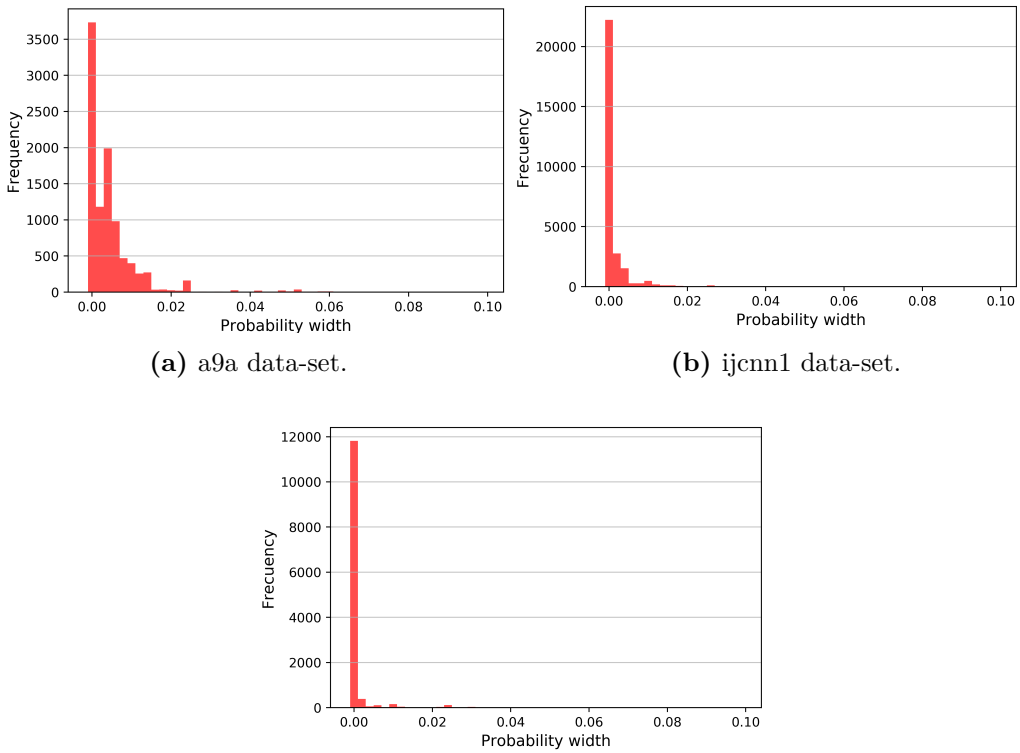
(e) webspam data-set.

(f) higgs data-set.

Figure 5.3: The histogram plots over the probability interval width for IVAP-WEB demonstrates that most intervals are close to zero. Thus, IVAP-WEB retains efficiency.

capability of capturing non-linearity through the use of kernels, is one of the greatest results in the field of machine learning in the last couple of decades.

Using the predictive accuracy of SVM within the framework of VAP, however, requires tremendous computing power, and more often than not, this resource is



(a) a9a data-set.

(b) ijcnn1 data-set.

(c) w8a data-set.

Figure 5.4: The histogram plots over the probability interval width obtained by VAP. Only a9a, ijcnn1 and w8a converged in our time budget.

simply not available. We believe that it is not necessary to modify an algorithm in order to make it more computationally efficient. This general hypothesis was tested with the protocol we have proposed in this chapter, IVAP-WEB. We have observed that the IVAP-WEB indeed is able to retain good calibration thanks to the strong theoretical guarantees of coresets. Quite interestingly, these guarantees seem to transfer well into practical applications for the VAP framework, and we demonstrated that, when the coreset is replaced with an arbitrary uniform random sample, calibration is damaged in unpredictable fashion. Thus, our coreset-based protocol sits at the sweet spot between computational efficiency and predictive accuracy.

A point that is worth highlighting here is that for moderate-sized data-sets such as Covertype, Ijcnn1 and Higgs, enjoying probability-type results from SVM with VAP proved to be computationally exhaustive and thus infeasible. Therefore, in these scenarios, IVAP-WEB seems to be the best viable answer to VAP's

computational dilemma.

I believe that at the end of the century the use of words and general educated opinion will have altered so much that one will be able to speak of machines thinking without expecting to be contradicted ...

— Alan Turing [116]

6 Efficient Data Summarisation based on Coresets

6.1 Introduction

In the previous chapters, we have shown that the use of the algorithmic framework of coresets can bring substantial computational gains to Conformal Prediction and to Venn-Abers Predictors. Thus, no matter how big the input data become, we can rely on a coreset construction algorithm to *approximately correctly* reduce its size and use the resulting compressed data to efficiently obtain prediction sets that retain both the validity and efficiency, for CP, and well-calibrated probability regions for VAP. Of course, the last statement holds, at least in our investigation, for the ML methods of LR and SVM.

At many points in our previous discussions, we have mentioned that there are more than one approach for constructing coresets. In chapter 4, we have used two algorithms: the sensitivity-based probabilistic RCA and the geometric-decomposition approach of enclosing balls through the AVM method. Fixing our focus on the former, we can notice, from its definition in Section 3.5.2, that RCA requires a clustering of the input data in order to determine the importance of every input example for the LR problem. This might seem strange at first, as clustering is not the main learning problem we are trying to solve in the first place. In fact,

clustering is known to be a computationally challenging problem on its own right. Furthermore, clustering is far more computationally expensive than LR ([117]).

It turns out that the above observation can effectively point us to the following question: *how can a computationally hard problem be used to accelerate the convergence of an easier computational problem such as logistic regression?* The short answer is that, in order to compute a coreset for LR, we are not required to provide an optimal clustering of the input data; a *rough* sub-optimal solution will suffice for computing the importance of input points. However, this question does lead us to an important issue: in some important scenarios, even the computation of a sub-optimal clustering of the input data can substantially slow the data compression process.

The main topic of this chapter thus is to analyse the computational stress caused by clustering in the computation of coresets for logistic regression. We will show that the RCA method can paradoxically slow the learning of a logistic regression classifier, defeating the purpose of using coresets to accelerate the learning process. The most important contribution of this chapter is the discovery that it is not necessary to cluster the full input data in order to obtain high-quality coresets [1]. Based on this fundamental claim, we will define two general procedures to save computing time in the coreset constructions: the *ACvS* procedure, which is a direct implementation of the research discovery mentioned; and the *RDSF* procedure, which builds upon ACvS and attempts to use some of the computing time saved to obtain extra information from the input data.

Finally, for this chapter, we leave the framework of Reliable Machine Learning on the side, since the main computational issue is exclusively related to the construction of coresets. This means, in principle, that the procedures proposed here can also be applied to coreset-based inductive conformal prediction, proposed in Chapter 4, and to inductive Venn-Abers predictors with enclosing balls, proposed in chapter 5.

[1]This chapter is based on two of our works: 1) the original work published in the 9th International Conference on Data Science, Technology and Applications, Paris 2020, where the article received the Best Student Paper award. See [118]; 2) An extended version of the original article that is currently under review for publication in Springer's Communications in Computer and Information Science (CCIS) series.

6.2 Computational Setting and Problem

As previously mentioned, we leave aside the methods from reliable machine learning in order to focus specifically on the problem of constructing coresets for logistic regression. To have a clear picture of the problem of logistic regression and of the RCA method for constructing coresets, their definitions can be found in Chapter 2 and Chapter 3, respectively.

The computations we will discuss assume an offline setting, where we have the whole input data $\mathcal{D}$ in advance *i.e.* as opposed to the online setting in which we gradually receive the input examples as part of an incoming sequence.

Our contributions rely on the fact that, in order to accelerate the LR learning process, the framework of coresets allows us to reduce the size of the input data $\mathcal{D}$ to a small coreset $\mathcal{C}$, where $|\mathcal{C}| \ll |\mathcal{D}|$ holds. After $\mathcal{C}$ is obtained, $|\mathcal{D}|$ can be safely disregarded and we can proceed to learn a LR classifier on the coreset only. Notice that, by definition, the theory of coresets guarantees that, even though model-training occurs over smaller data, the learning performance of the resulting LR classifier will be approximately retained; this implies that the predictions made with a classifier learned from a coreset are protected from arbitrary quality loss.

RCA takes as input the data $\mathcal{D}$ and a set of k-centres $Q_{\mathcal{D}}$, the latter is the result of applying a clustering algorithm to $\mathcal{D}$, and outputs a coreset $\mathcal{C}$. Even though this works well in principle, we will see in the next section that the computation of the k-centres in $Q_{\mathcal{D}}$ can be computationally infeasible for a coreset algorithm. The problem of finding a clustering for a set of points is known to be computationally hard [27]. This is why approximation and data reduction techniques are popular approaches for accelerating clustering algorithms. In fact, the paradigm of coresets has seen great success in the task of approximating solutions for clustering problems (see [67], [70], [86], [69] and [98]).

The *sensitivity framework* for defining coresets, presented in Chapter 3, requires a sub-optimal clustering of the input data $\mathcal{D}$ in order to compute the sensitivity for each point in the set. This requirement transfers to the RCA algorithm. If we assume a Bayesian setting as in[78], the time necessary for clustering $\mathcal{D}$ is dominated

by the cost of running posterior inference algorithms. However, if we remove the burden of posterior inference and consider an optimisation setting, such the one we previously defined, then the situation is very different.

Figure 6.1 sheds light on the above phenomenon; here,we can see the time spent on finding a clustering compared to all the other steps taken by RCA to construct a coreset, namely: *sensitivity computation* and *sampling*. The time spent on training the LR model on the coreset is included as well.

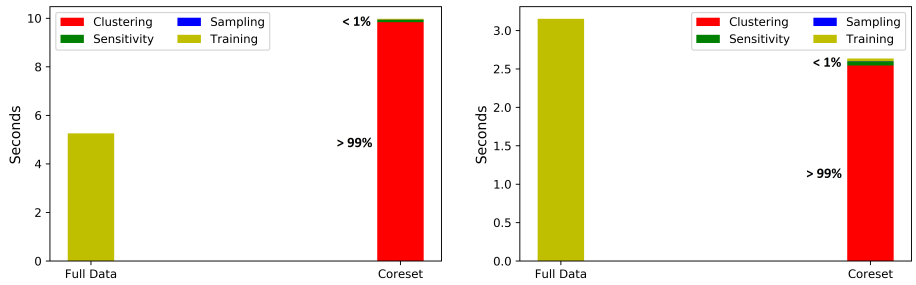
It is easy to see that applying a clustering algorithm, even to get a sub-optimal solution as done here, can be dangerously impractical for computing a coreset in the optimisation setting as it severely increases the overall coreset-construction time. Even worse, constructing the coreset is slower than learning directly from $\mathcal{D}$, defeating the purpose of using the coreset as an acceleration technique.

Another interesting point that can be mentioned here is that any algorithm used for computing coresets must give us summaries of data that are both quickly-computable and quality-preserving. Disregarding one of these objectives makes the task of summarising the input data much easier, but ultimately the summaries are less effective; that is, we can simply take a uniform random sample from our input data, which can be achieved in constant time; no compression algorithm will finish faster, but we will suffer a quality loss that is out of our control. Similarly, on the other side of the spectrum, we can just avoid performing any data compression at all, and we will for sure be able to obtain a very high-quality solution; but we should be ready to deal with lots of computational stress, storage overflows and even loss of random access[2].

Given the above discussion, we can ask the following research question: "Can we still benefit from good coreset acceleration in the optimisation setting?"

We give an affirmative answer to this question through two approaches described in the next section. The key ingredient for both methods is the use of *Uniform Random Sampling* (URS) alongside coresets.

[2]we say we loose random access when the input data is so large that accessing some parts of the data is more expensive, computationally speaking, than accessing other parts.



(a) Webspam **(b)** Covertype

Figure 6.1: Comparison of the computing time needed for constructing LR coresets for the Webspam and Covertype datasets. The clustering process consumed the majority of the processing time.

6.3 The Computational Remedy

In this section, we propose two different procedures for efficiently computing coresets for Logistic Regression in the optimisation setting. Both approaches are similar in the following sense: they both make use of uniform random sampling for speeding up coresets computation. URS is the most straightforward and simple way to reduce the size of input dataset by picking as many points as permissible uniformly at random. This is not the first time that the concept of URS comes up alongside coresets; in fact, URS can be seen as a naive approach for computing coresets and it is the main motivation for deriving a more sophisticated sampling approach [86]. In the following procedures, however, we see URS as a complement to coresets, not as an alternative to them, which is usually the case in coresets works.

6.3.1 Accelerated Clustering via Sampling

Our first procedure, **A**ccelerated **C**lustering **v**ia **S**ampling (ACvS), uses a straightforward application of URS. The procedure is described in Algorithm 9. First, we extract b input points from $\mathcal{D}$ and put them into a new set S. We require that $b \ll n$, where $n := |\mathcal{D}|$. Then, we cluster S to obtain k cluster centres, namely Q_S with $|Q_S| := k$. We finally run the RCA algorithm using Q_S as input and compute a coreset. Since we have that $|S| \ll |\mathcal{D}|$, obtaining Q_S is substantially faster than obtaining $Q_{\mathcal{D}}$. Notice that the coreset algorithm RCA is parameterised by the coreset size M. As a simple example, say we have a large dataset that we

Algorithm 9: ACvS procedure.

Input: RCA: coreset algorithm, $\mathcal{D}$: input data, A: a clustering algortithm, k: number of cluster centres, $b \ll |\mathcal{D}|$: uniform random sample size

Output: $\mathcal{C}$: coreset

```
S ← ∅;
B ← |S|;
while B < b do
    s ← SamplePoint(D) // Sample without replacement
    S ← S ∪ {s} // Put s in S
end
Q_S ← A(S,k) // Run Clustering algorithm on S
C ← RCA_M(D, Q_S) // Run Coreset Algorithm
return C
```

would like to classify using logistic regression, and to do it quite efficiently, we decide to compress the input data into a coreset. The standard coreset algorithm for LR dictates that we have to find a clustering of our dataset as the very first step; then we use the clustering to compute the sensitivity of each point; the next step is to sample points according to their sensitivity and to put them in the coreset, and finally we compute the weight for each point in the coreset. What ACvS proposes is: instead of computing the clustering of the full input data, compute the clustering of a very small set of uniform random samples of it, then proceed as the standard coreset algorithm dictates.

We designed this procedure based on the following observation: uniform random sampling can provide unbiased estimation for many cost functions that are additively decomposable into non-negative functions. We prove this fact below.

Let $\mathcal{D}$ be a set of points and let $n := |\mathcal{D}|$; also, let Q be a *query* whose cost value we are interested in computing. We can define a cost function which is decomposable into non-negative functions as $cost(\mathcal{D}, Q) := \frac{1}{n} \sum_{x \in \mathcal{D}} f_Q(x)$.

Many machine learning algorithms can be cast to this form. Here, we are mainly concerned about k-Means clustering, where Q is a set of k points and $f_Q(x) := min_{q \in Q} ||x - q||_2^2$. More details on k-means can be found in Section 2.4.1.

Let us now take a random uniform sample $S \subset \mathcal{D}$ with $m := |S|$, and define the cost of query Q with respect to S as $cost(S, Q) := \frac{1}{m} \sum_{x \in S} f_Q(x)$. To show

that $cost(S, Q)$ is an unbiased estimator of $cost(\mathcal{D}, Q)$ we need to prove that $\mathbb{E}[cost(S, Q)] = cost(\mathcal{D}, Q)$

Claim. $\mathbb{E}[cost(S, Q)] = cost(\mathcal{D}, Q)$

Proof. By definition, we have that $cost(S, Q) := \frac{1}{m} \sum_{x \in S} f_Q(x)$. Expanding this, we get

$$\mathbb{E}[cost(S, Q)] = \mathbb{E}[\frac{1}{m} \sum_{x \in S} f_Q(x)] \tag{6.1}$$

The crucial step now is to compute the above expectation. To do this, it is useful to construct the set $\mathbb{S}$ that contains all the possible subsets S in $\mathcal{D}$. The number of such subsets has to be $\binom{n}{m}$, which implies $|\mathbb{S}| := \binom{n}{m}$. Then, computing the expectation over $\mathcal{S}$ and re-arranging some terms we get

$$\frac{1}{\binom{n}{m}} \frac{1}{m} \sum_{S \in \mathbb{S}} \sum_{x \in S} f_Q(x) \tag{6.2}$$

Next, we get rid of the double summation as follows: we count the number of times that $f_Q(x)$ is computed. By disregarding overlapping computation of $f_Q(x)$ due to the fact that a point x can belong to multiple subsets $S \in \mathbb{S}$, we can quickly obtain that the count has to be $\binom{(n-1)}{(m-1)}$. Then, we can write (6.2) as

$$\frac{1}{\binom{n}{m}} \frac{1}{m} \binom{(n-1)}{(m-1)} \sum_{x \in \mathcal{D}} f_Q(x) \tag{6.3}$$

Notice that now we have a summation that goes over our original set $\mathcal{D}$. Finally, by simplifying the factors on the left of the summation we have

$$\frac{1}{n} \sum_{x \in \mathcal{D}} f_Q(x) = cost(\mathcal{D}, Q) \tag{6.4}$$

which concludes the proof. □

The imminent research question here is: how does using the ACvS procedure affect the performance of the resulting coreset? We shall show in Section 6.4 that the answer is indeed positive.

6.3.2 Regressed Data Summarisation Framework

The second method builds on the previous one and it gives us at least two important benefits on top of the acceleration benefits given by ACvS:

- *sensitivity interpretability:* it unveils an existing (not-obvious) linear relationship between input points and their sensitivity scores.
- *instant sensitivity-assignment capability:* apart from giving us a summary of the input data, it gives as a trained regressor capable of assigning sensitivity scores *instantly* to new unseen points.

Before presenting the method, however, it is useful to remember the following: the RCA algorithm (Algorithm 3) implements the sensitivity framework and hence it relies on computing the sensitivity, a measure of importance, for each input point.

We call our second procedure *Regressed Data Summarisation Framework* (RDSF). We can use this framework to (i) accelerate a sensitivity-based coreset algorithm; (ii) unveil information on how data points relate to their sensitivity scores; (iii) obtain a regression model that can potentially assign sensitivity scores to new data points.

The full procedure is shown in Algorithm 10: RDSF starts by using ACvS to accelerate the clustering phase. The next step is to separate the the input data $\mathcal{D}$ in two sets: S, the small URS picked during ACvS, and R, all the points in $\mathcal{D}$ that are not in S. The main step in RDSF starts at line 12: using the clustering obtained in the ACvS phase, Q_S, we compute the sensitivity scores *only* for the points in S and place them in a predefined set Y. A linear regression problem is then solved using the points in S as feature vectors and their corresponding sensitivity scores in Y as targets. Thus, RDSF poses the problem of summarising data as the problem of 'learning' the sensitivity of the input points. The result of such learning process is a trained regressor ϕ and RDSF uses it to predict the sensitivities of all the points in R. Hence, RDSF uses S as training set and R as test set for the inner regression problem.

Algorithm 10: The Regressed Data Summarisation Framework (RDSF) uses a coreset construction to produce coreset-based summaries of data.

Input: $\mathcal{D}$: input data, A: clustering algortithm, k: number of cluster centres, $b \ll |\mathcal{D}|$: uniform random sample size, M: coreset size
Output: $\tilde{\mathcal{C}}$: Summarised Version of $\mathcal{D}$, ϕ: Trained Regressor

1 initialise;
2 $S \leftarrow \emptyset$;
3 $N \leftarrow |\mathcal{D}|$;
4 **while** $|S| < b$ **do**
5 $\quad s \leftarrow$ `SamplePoint(`$\mathcal{D}$`)` `// Sample without replacement`
6 $\quad S \leftarrow S \cup \{s\}$ `// Put` s `in` S
7 **end**
8 $Q_S \leftarrow$ `A(`S`,`k`)` `// Run Clustering algorithm on` S
9 $R \leftarrow \mathcal{D} \setminus S$;
10 $Y \leftarrow \emptyset$;
11 **for** $i = 1, 2, ..., b$ **do**
12 $\quad m_n \leftarrow$ `Sensitivity(`s_i, Q_S`)` `// Compute the sensitivity of point` $s_i \in S$
13 $\quad Y \leftarrow Y \cup \{m_n\}$;
14 **end**
15 $\hat{Y}, \phi \leftarrow$ `PredictSen(`S, Y, R`)` `// Train regressor on` S `and` Y`, predict sensitivity for each` $r \in R$
16 $Y \leftarrow Y \cup \hat{Y}$;
17 $\bar{m}_N \leftarrow \frac{1}{N} \sum_{y \in Y} y$;
18 **for** $i = 1, 2, ..., n$ **do**
19 $\quad p_i = \frac{m_i}{n\bar{m}_n}$; `// compute importance weight for each point`
20 **end**
21 $(K_1, K_2, ..., K_n) \sim$ `Multi(`M`,`$(p_i)_{i=1}^n$`)` ; `// sample coreset points`
22 **for** $i = 1, 2, ..., n$ **do**
23 $\quad w_n \leftarrow \frac{K_n}{p_n M}$; `// calculate the weight for each coreset point`
24 **end**
25 $\tilde{\mathcal{C}} \leftarrow \{(w_i, x_i, y_i) | w_i > 0\}$;
26 **return** $\tilde{\mathcal{C}}$,ϕ

We finally see that after merging the computed and predicted sensitivities of S and R (line 16 in Algorithm 10), respectively, RDSF executes the same steps as RCA *i.e.* compute the mean sensitivity (line 19), sample the points that will be in the summary (line 22) and compute the weights (line 23).

6.4 Experiments and Results

6.4.1 Strategy

In this section, evaluation results for the two procedures ACvS and RDSF are presented. As before, we tested our procedures on the datasets in our master table for data, presented in Chapter 2.

In order to test the effectiveness of the proposed methods, the experiments compare the following five procedures:

- **Full:** no coreset or summarisation technique is used. That is, we simply train a LR model on the entire training set and predict the labels for the instances in the test set.

- **RCA:** we obtain a clustering of the input data and run RCA (Algorithm 3) to obtain a coreset. We then train a LR model on the coreset to predict the labels for the test instances.

- **ACvS:** we use the procedure 'Accelerated Clustering via Sampling', described in Algorithm 9, to accelerate the coreset computation. Once we have obtained the coreset in accelerated fashion, we proceed to learn a LR classifier over it.

- **RDSF:** summaries of data are generated via the 'Regressed Data Summarisation Framework', described in Algorithm 10. Hence, we compute sensitivity scores only for a handful of instances in the training set. Then, we train a regressor to predict the sensitivity scores for the remaining of the training instances. We sample points according to the sensitivities, compute their weights and return the data summary. We then proceed as in the previous coreset-based procedures.

- **URS:** for the sake of completeness, we include 'Uniform Random Sampling' as a baseline for reducing the volume of input data; we simply pick the required input points uniformly at random and then train a LR classifier over them.

Our evaluation pipeline can be described as follows: for each of the above approaches, and for each dataset in Table 2.1, we take the below steps:

1. **data shuffling:** we randomly mix up all the available data.

2. **data splitting:** we select 50% of the data as training set and leave the rest for testing purposes. Thus, the training set is the input data $\mathcal{D}$.

3. **data compression:** we proceed to compress the input data. As previously mentioned, the *RCA* approach computes a coreset without any acceleration, *ACvS* computes a coreset by using RCA with an accelerated clustering phase, *RDSF* compresses the input data into a small summary data in an accelerated fashion, and *URS* performs a naive compression by taking an uniform random sample of the input data. *Full Data* is the only approach that does not perform any compression on the input data.

4. **model training:** we train a Logistic Regression classifier on the data obtained in the previous step. *RCA*, *ACvS*, *RDSF* and *URS* produce a reduced version of the input data while *Full Data* trains the classifier on the full input data $\mathcal{D}$.

5. **output assessment:** we finally use the trained Logistic Regression classifiers to predict the labels in the test set and apply our performance metrics, detailed in Section 6.4.2.

The above steps are repeated 10 times for each of the five different approaches. Hence, the results we will see in the next sections are averaged ones. One important observation concerning the RDSF method is that the results in this chapter were generated using the classic *Ordinary Least Squares* (OLS) as regressor. However, any regressor could be used in order to make the sensitivity predictions. Appendix D presents results obtained by using RDSF with regressors such as Ridge Regression ([119]), Lasso Regression ([120]) and Elastic-Net Regression ([121]).

All of our programs were written in Python. The method used for clustering the input data is the well-known k-Means algorithm; and for RDSF, linear regression is

used to learn the sensitivities of input points. Both k-Means and linear regression were formally defined in Chapter 2.

6.4.2 Metrics

The empirical performance of coresets has mainly remained a grey area in the past years. We apply the following performance metrics in order to shed light on, first, the performance of coresets in general, second, the performance of the methods proposed in this chapter. We shall consider the following five performance metrics:

1. **Computing time (in seconds):** here we measure acceleration in seconds. To make a more meaningful analysis, we further break down time into 5 different stages :

   (a) **Clustering:** the time needed to obtain the k-centres for coreset-based approaches.

   (b) **Sensitivity:** the time required to compute the sensitivity score for each input point.

   (c) **Regression:** the time needed for training a regressor in order to predict the sensitivity scores for input points. The prediction time is also taken into account.

   (d) **Sampling:** the time required to sample input points.

   (e) **Training:** the time required for leaning an LR classifier.

   Notice that the *Training phase* is the only one present in all our approaches. Hence, for example, the *coreset* approach does not learn any regressor and thus it is assigned 0 seconds for that phase. The URS approach does not perform any clustering or sensitivity computation, hence those phases get 0 seconds for this method, etc.

2. **Classification Accuracy:** this measure is given by the percentage of correctly classified test examples. It is commonly used as the baseline metric for measuring performance in a supervised-learning setting.

3. **Area Under the Precision & Recall Curve (PREC/REC):** Precision is defined as $\frac{TP}{TP+FP}$ and Recall can be computed as $\frac{TP}{TP+FN}$ [122], where TP,FP and FN stand for the *True Positives*, *False Positives* and *False Negatives* achieved by a binary classifier, respectively. The curve is obtained by putting the Recall on the x-axis and the Precision on the y-axis. Once the curve has been generated, the area under the curve can be calculated in the interval between 0 and 1. The greater the area, the better the performance.

4. **F1 Score:** is the harmonic average of precision and recall [123] and hence can be computed as $F_1 := 2\frac{PR}{R+P}$, where P is Precision and R is recall [123]. The greater the value, the better the classifier's performance.

5. **Area Under the ROC Curve (AUROC):** provides an aggregate measure of performance across all possible classification thresholds. The curve can be computed by placing the *False Positives Rate* (FPR) on the x-axis and the *True Positives Rate* (TPR) on the y-axis, with FPR $:= \frac{FP}{FP+TN}$ and TPR $:= \frac{TP}{TP+FN}$; here, once more, TP,FP and FN stand for the *True Positives*, *False Positives* and *False Negatives* achieved by any binary classifier, respectively. Similar to the area under the precision and recall curve, AUROC is obtained from the curve.

6.4.3 Acceleration via ACvS and RDSF

We categorise our results according to the five different metrics we just described. Notice that their values are shown as functions of the size of the summaries used for training the LR classifier. Thus, if we look at Figures 6.3, 6.5, 6.4 and 6.6, we can see that summary sizes on the x-axes correspond to very small percentages of the training set. Specifically, for the Higgs dataset, which is the largest one, the summary sizes are 0.005 %, 0.03 %, 0.06 % and 0.1 % of the input data. For Webspam and Covertype, which are smaller then Higgs, we show results with summary sizes of 0.05 %, 0.1 % , 0.3 %, 0.6 % and 1 % of the total input data. Finally, for w8a and ijcnn1, which are the smaller datasets, the sizes shown are 1 %,

3 % , 6 %, and 10 %. The reason why there are different summary sizes for some of our datasets has to do with the difference in size across datasets. For example, computing a summary of 0.005 % of the w8a or the ijcnn1 datasets is unfeasible since these datasets are not very large and hence it is very likely to end up with a extremely small summary of data that only contains points of one class. In other words: the larger the dataset, the more we can compress it.

Computing Time

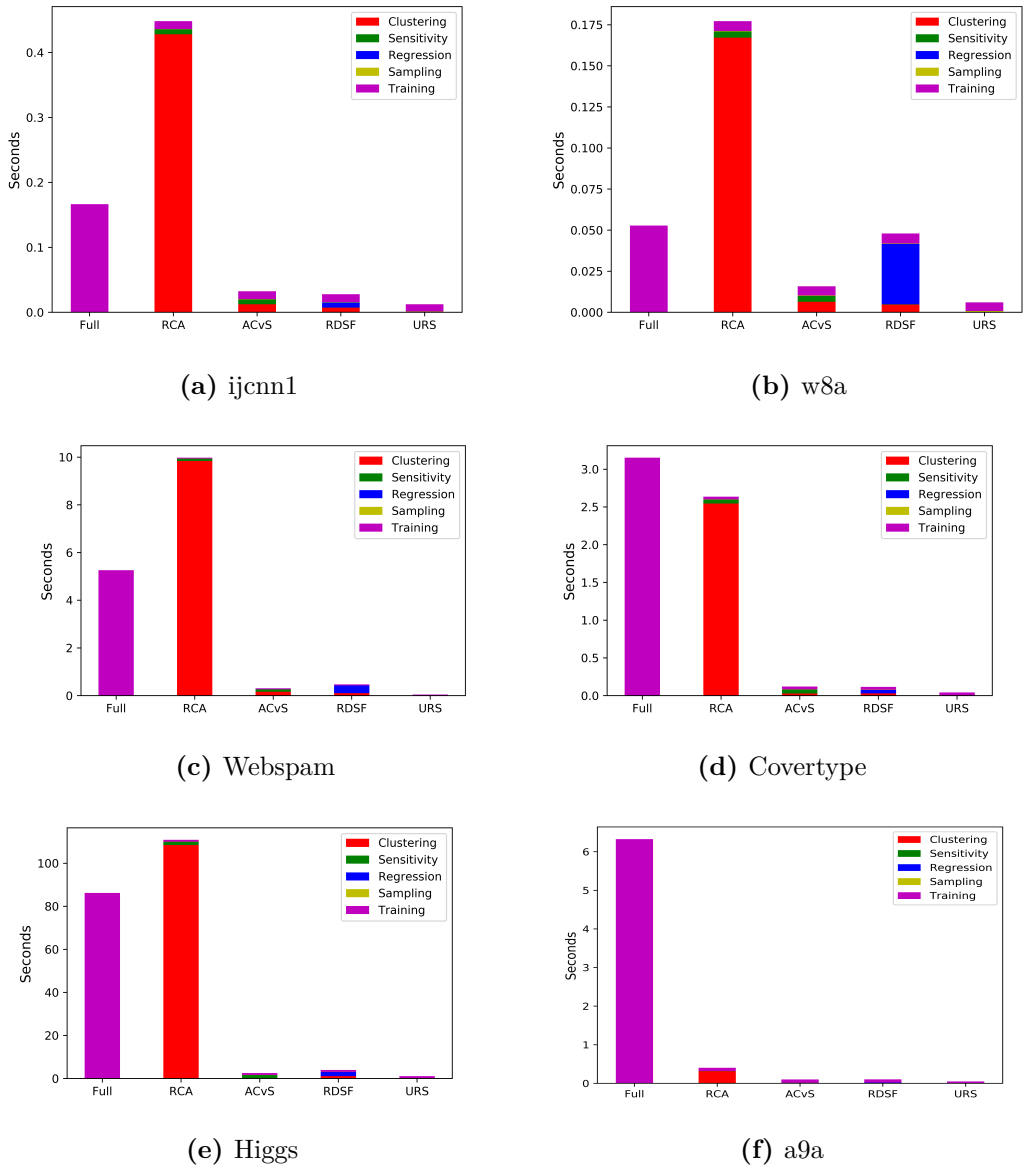
Figure 6.2 summarises our results in terms of computing time. We show in the stacked-bars plots the time spent for each *phase* of the different approaches.

We can clearly see that the RCA approach, which clusters the full input data in order to constructs coresets , is not suitable for the optimisation setting we are considering. Specifically, with respect to the Full method, we notice that for the Covertype dataset, the coreset approach gives a modest acceleration of approximately 1.2 times. The situation becomes more severe for the Webspam, ijcnn1, w8a and Higgs datasets, where using the traditional coreset approach incurs in a learning process which is about 1.9, 2.6, 1.8 and 1.3 times slower than not using coreset at all, respectively.

Hence, by removing the bottleneck produced by the clustering phase, our two proposed methods show that we can still benefit from coreset acceleration to solve our particular problem; that is, our methods take substantially shorter computing time when compared to the *RCA* approach. In particular, and with respect to Full Data approach, our approach *ACvS* achieves a minimum acceleration of 3.5 times (w8a) and a maximum acceleration of 34 times (Higgs) across our datasets. Regarding *RDSF*, the minimum acceleration obtained was 1.15 times (w8a) and the maximum was 27 times (Covertype).

Notice that our accelerated methods are only beaten by the naive URS method, which should most certainly be the fastest approach.

Finally, notice that the *RDSF* approach is slightly more expensive than *ACvS*. This is the computing price we pay for obtaining more information *i.e.* *RDSF*



(a) ijcnn1 **(b)** w8a

(c) Webspam **(d)** Covertype

(e) Higgs **(f)** a9a

Figure 6.2: Comparison of the computing time of different methods.

outputs a trained regressor that can immediately assign sensitivities to new unseen data points; this can prove extremely useful in settings when learning should be done *on the fly*. According to our evaluations, RDSF's time can be improved by reducing the number of points used for training the underlying regression algorithm. In general, using 1 % of the training set for training the regressor worked well; however, depending on the data, this could be reduced (or increased) in order to achieve better computing time (learning quality).

The natural follow-up question is whether our methods' resulting classifiers perform well. We address this question in great detail in the following sections.

Accuracy

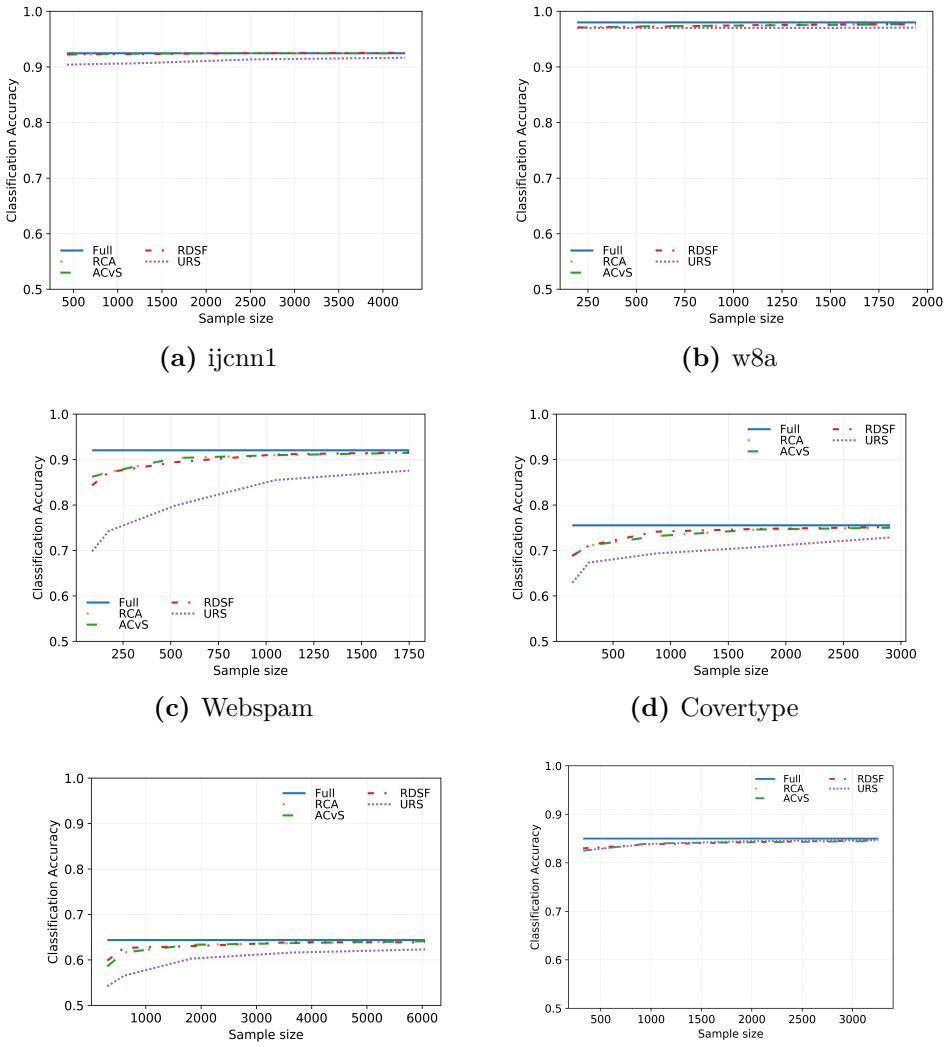
We first look into the baseline metric for measuring the success of a classifier: the accuracy. Figure 6.3 shows how the accuracy of the methods changes as the sample sizes increase on different datasets. To recall, each of the different methods considered relies on reducing the input data via a coreset-based compression or a random uniform sample, as described in Section 6.4.1. Hence, we here report the different accuracy scores obtained by training our LR classifier on different samples sizes. As reference, we also include the accuracy of the *Full Data* method as a straight line.

The first observation is that all the methods perform better as the sample sizes increase. Quite surprisingly, we see that in all cases, without exceptions, the ACvS approach achieves the exact same accuracy that the RCA approach achieves, for all sample sizes. Hence, for coresets, clustering over a sub-sample of the input data does not seem to deteriorate the rate of correct predictions of the resulting classifiers, and greatly accelerates the overall coreset computation, as we could appreciate in the previous section.

We also see that the RDSF approach performs as good as the rest of the coreset-based approaches, with accuracy never lower than the baseline approach (URS). It is generally expected that coresets outperform the URS approach in most situations; and RDSF summaries, even without being strictly a coreset [3], show this behaviour.

Finally, we see that, as sample sizes increase, the gap between coreset performance and URS performance reduces *i.e.* they both get closer and closer to the Full Data approach. A particularly interesting case is that of w8a, which shows a very similar performance for all the methods. This could be an indicator that points in the dataset contribute *almost* equally to the learning problem considered: LR, in this case.

[3] we carefully distinguish between a coreset and a coreset-based summary. The former requires a theoretical proof on the quality loss.



(a) ijcnn1 **(b)** w8a

(c) Webspam **(d)** Covertype

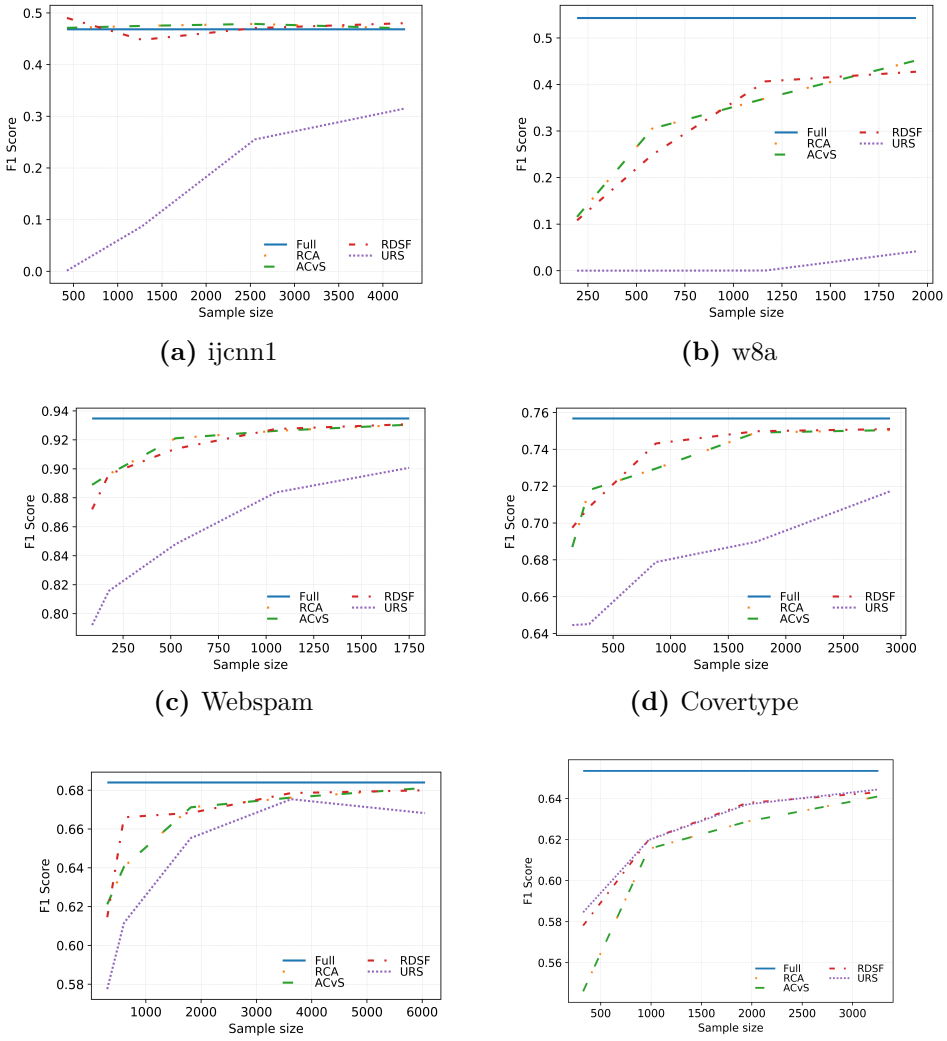
(e) Higgs **(f)** a9a

Figure 6.3: Comparison of the prediction accuracy of different methods.

F1 Score

We now present the results of applying the F1 score metric to our LR classifiers for different sample sizes, see Figure 6.4.

The first observation we make is that, similar to accuracy results, ACvS and RCA obtain exactly the same scores, and RDSF remains competitive against them. Furthermore, it is fair to say that for the Covertype and Higgs datasets, RDSF has preferable performance compared to the other compression approaches. Hence, we once more see that the advantages of coresets are available in our setting as long as we carefully accelerate the underlying algorithm.



(a) ijcnn1 **(b)** w8a

(c) Webspam **(d)** Covertype

(e) Higgs **(f)** a9a

Figure 6.4: Comparison of F1 score of different methods.

Our experiments reveal that the performance gap between URS and the rest of the approaches becomes even greater for the F1 Score; showing that classifiers trained over coresets are more useful and informative than the ones trained over uniformly randomly selected samples. Furthermore, if we look at F1 score for ijcnn1 (see Figure 6.6a) we can have a glimpse of an interesting phenomenon: we actually obtain better results using coresets, and hence *less* data, than using the full dataset. We leave this as an open problem for future exploration.

AUROC

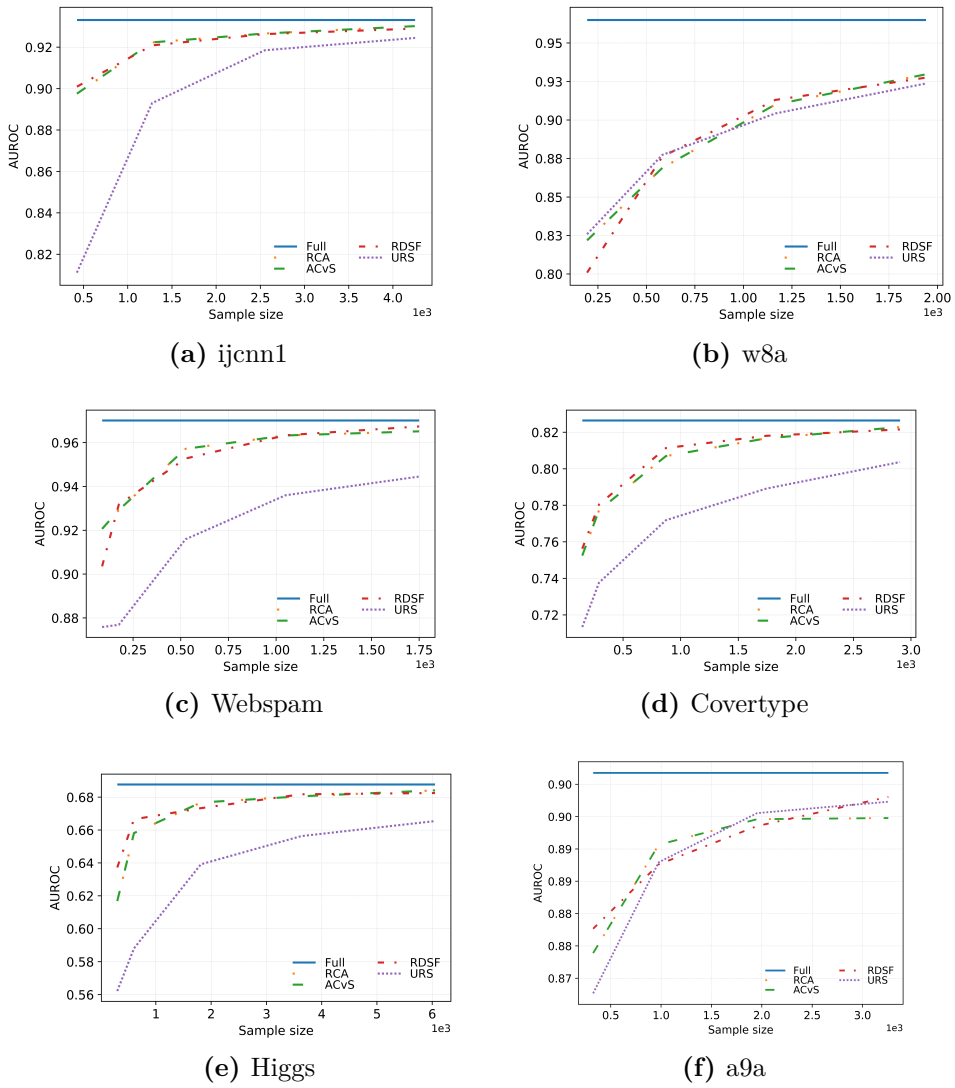
We now present the AUROC score for each of our 5 approaches. Figure 6.5 shows comparison of the AUROC scores for all methods. The picture is similar to that of F1-score and Accuracy: coreset-based approaches consistently outperform URS. We see indeed that ACvS and RDSF perform competitively traditional coreset approach, achieving their performance in substantially less computing time than coresets (see Section 6.4.3). An intriguing case is that of w8a, which shows that URS is actually slightly better than coreset approaches for small sample sizes. As we previously mentioned, it is highly probably that this is an indication that the input points in the dataset are not very different in terms of their contribution to the LR objective function; or, it could also be the case that the sensitivity distribution, as computed by the coreset-based methods, does not fully account for the structure of the data.

Precision & Recall

Finally, we present the results concerning the precision and recall score. The behaviour here is similar to that of AUROC. The performance of w8a seems to be different from the rest of the data once more: we see that, for very small sample sizes, URS is even slightly better that the coreset and coreset-based approaches. As the sample sizes increase, the latter outperform the former, although not by much. As we previously mentioned, this could mean that input points in w8a are more or similar for LR and hence we cannot strictly distinguish between redundant and important points. We also see that ACvS gets exactly the same scores when compared to its non-accelerated version; which shows that clustering over the whole input data is not necessary *i.e.* clustering over a small uniform random sample is sufficient.

Summary of Results

This section provides detailed performance information of our procedures, alongside the rest of the methods, on the six data-sets in Table 2.1. Specifically, Tables 6.1, 6.2, 6.3, 6.4 and 6.5 show the different metric scores obtained over our set of datasets.



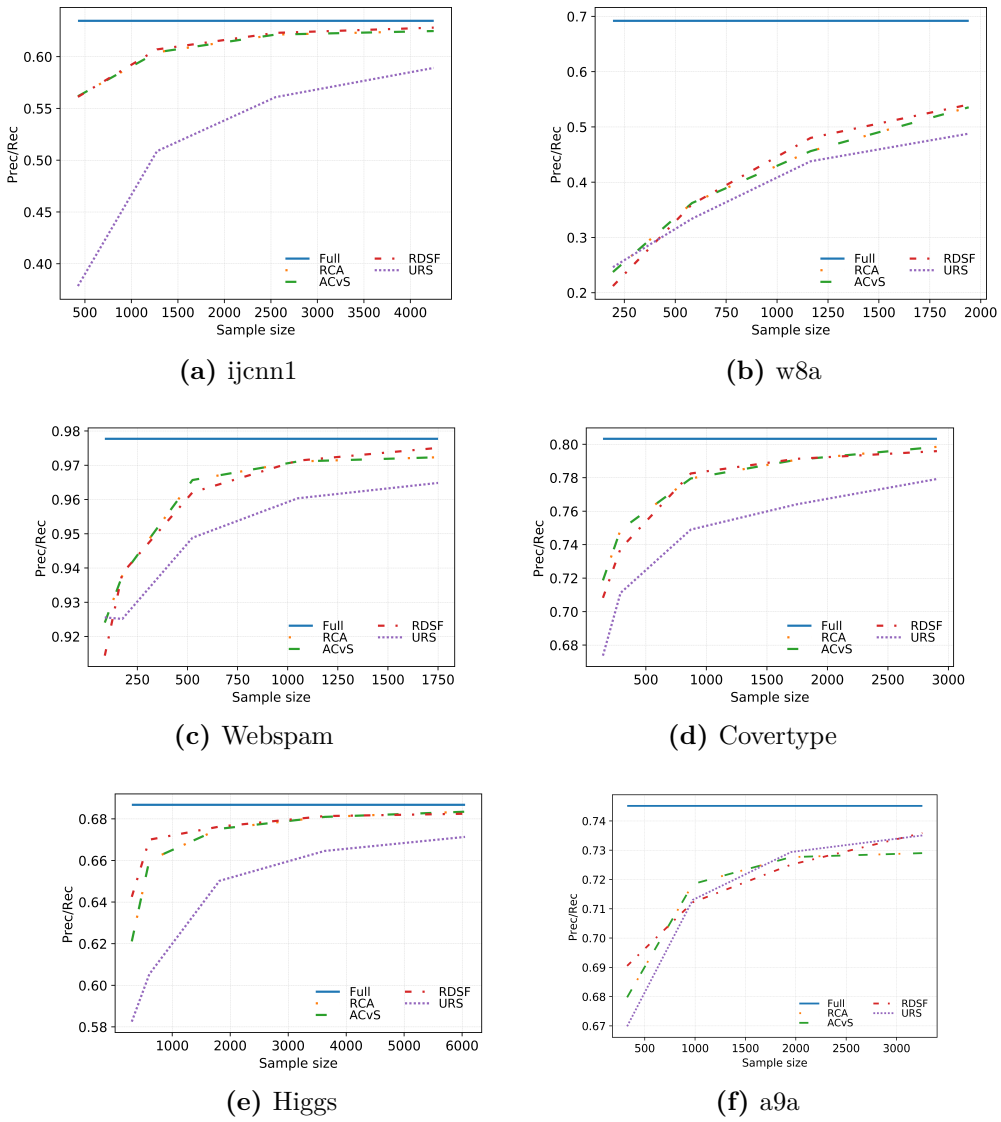
(a) ijcnn1 **(b)** w8a

(c) Webspam **(d)** Covertype

(e) Higgs **(f)** a9a

Figure 6.5: Comparison of the area under the ROC curve of different methods.

Overall, and as shown in the plots in previous sub-sections, the empirical results demonstrated that ACvS and RDSF do provide meaningful acceleration to the traditional Coreset approach, while maintaining competitive performance, in all datasets considered. One important observation is that, even though our procedures give good speed-ups, the performance of coresets are dataset dependent. If the structure of the data can be captured correctly by doing an uniform random sample, then coreset performance should not be expected to be very different than URS. We can see an example of this in w8a, where coresets in general do not give very meaningful improvement. On the other hand, for the rest of the datasets, we



(a) ijcnn1 **(b)** w8a

(c) Webspam **(d)** Covertype

(e) Higgs **(f)** a9a

Figure 6.6: Comparison of the area under the precision/recall curve of different methods.

can indeed see how compressing the input data in more involved fashion gives a significant improvement over the naive URS.

6.5 Chapter Summary

As modern ever-growing sets of data overshadow our computing resources, scaling up machine learning algorithm is not a trivial task. The most direct algorithmic approach is to write new learning algorithms that overcome the inefficiencies of their old counterparts. A less direct approach consists in using the well-known algorithms

Table 6.1: Performance comparison on the Covertype dataset where size is the percentage of training data.

Size (%)	Method	F1 score	ROC	Accuracy	Time (seconds)
0.05	Full	0.76	0.83	0.76	3.35
0.05	RCA	0.69	0.75	0.69	2.78
0.05	ACvS	0.69	0.75	0.69	0.12
0.05	RDSF	0.70	0.76	0.69	0.12
0.05	URS	0.65	0.71	0.63	0.04
0.3	Full	0.76	0.83	0.76	3.31
0.3	RCA	0.73	0.81	0.73	2.78
0.3	ACvS	0.73	0.81	0.73	0.13
0.3	RDSF	0.74	0.81	0.74	0.13
0.3	URS	0.68	0.77	0.69	0.05
1	Full	0.76	0.83	0.76	3.71
1	RCA	0.75	0.82	0.75	3.38
1	ACvS	0.75	0.82	0.75	0.18
1	RDSF	0.75	0.82	0.75	0.18
1	URS	0.72	0.80	0.73	0.07

we currently possess over a reduced version of their input data: the coreset. We have shown that for the optimisation setting, the algorithm for constructing coresets for the problem of Logistic Regression relies on a clustering phase that, more often than not, creates a bottleneck in the compression process.

To circumvent this, we proposed two methods that can ease this computational bottleneck: Accelerating Clustering via Sampling (ACvS) and Regressed Data Summarisation Framework (RDSF). Both methods achieved substantial overall learning acceleration while maintaining the performance accuracy of coresets. This implies that coresets can still be efficiently used to learn a logistic regression classifier in the optimisation setting.

Interestingly, we observed that, even though RCA needs a clustering of the input data, this can be relaxed in the practical sense. Furthermore, our calculations indi-

Table 6.2: Performance comparison on the Webspam dataset where size is the percentage of training data.

Size (%)	Method	F1 score	ROC	Accuracy	Time (seconds)
0.05	Full	0.92	0.94	0.97	5.39
0.05	RCA	0.86	0.89	0.92	10.15
0.05	ACvS	0.86	0.89	0.92	0.31
0.05	RDSF	0.84	0.87	0.90	0.49
0.05	URS	0.70	0.79	0.88	0.05
0.3	Full	0.92	0.94	0.97	6.54
0.3	RCA	0.90	0.92	0.96	13.40
0.3	ACvS	0.90	0.92	0.96	0.41
0.3	RDSF	0.89	0.91	0.95	0.64
0.3	URS	0.80	0.85	0.92	0.06
1	Full	0.92	0.94	0.97	6.35
1	RCA	0.92	0.93	0.97	13.44
1	ACvS	0.92	0.93	0.97	0.45
1	RDSF	0.92	0.93	0.97	0.68
1	URS	0.88	0.90	0.95	0.08

cate that RCA must be used with the clustering done over a small subset of the input data in the optimisation setting (*i.e.* the ACvS approach). Our empirical evaluations confirm that, by doing so, one will not be sacrificing learning performance.

With respect to RDSF, I believe this opens a new direction for coresets: we could pose the computation of data compression via coresets as solving a small-scale learning problem in order to solve a large-scale one. It is interesting to see that the sensitivities of input points can be explained by a simple linear regressor. Most importantly, we believe that this method could be of powerful use in the *online learning setting* [124]. This is because RDSF allows us to obtain a fully trained regressor capable of assigning sensitivity scores to new incoming data points. We leave as future work the use of these methods in different machine learning tasks.

Table 6.3: Performance comparison on the Higgs dataset where size is the percentage of training data.

Size (%)	**Method**	**F1 score**	**ROC**	**Accuracy**	**Time (seconds)**
0.005	Full	0.68	0.69	0.64	89.22
0.005	RCA	0.62	0.62	0.59	112.44
0.005	ACvS	0.62	0.62	0.59	2.61
0.005	RDSF	0.62	0.64	0.60	4.16
0.005	URS	0.58	0.56	0.54	1.12
0.03	Full	0.68	0.69	0.64	92.22
0.03	RCA	0.67	0.68	0.633	121.25
0.03	ACvS	0.67	0.68	0.63	2.70
0.03	RDSF	0.67	0.67	0.63	4.46
0.03	URS	0.66	0.64	0.60	1.20
0.1	Full	0.68	0.69	0.64	90.18
0.1	RCA	0.68	0.68	0.64	123.97
0.1	ACvS	0.68	0.68	0.64	2.61
0.1	RDSF	0.68	0.68	0.64	4.50
0.1	URS	0.69	0.67	0.62	1.29

Table 6.4: Comparison of coreset performance on w8a dataset where size is the percentage of training data.

Size (%)	**Method**	**F1 score**	**ROC**	**Accuracy**	**Time (seconds)**
1	Full	0.543	0.965	0.98	0.06
1	RCA	0.116	0.822	0.971	0.18
1	ACvS	0.116	0.822	0.971	0.02
1	RDSF	0.108	0.801	0.971	0.05
1	URS	0	0.826	0.97	0.007
3	Full	0.543	0.965	0.98	0.06
3	RCA	0.305	0.869	0.973	0.19
3	ACvS	0.305	0.869	0.97321	0.02
3	RDSF	0.249	0.876	0.97237	0.06
3	URS	0	0.877	0.97	0.01
6	Full	0.543	0.965	0.98	0.05
6	RCA	0.37	0.91	0.975	0.19
6	ACvS	0.37	0.91	0.975	0.02
6	RDSF	0.407	0.913	0.975	0.05
6	URS	0.0003	0.904	0.97	0.01

Table 6.5: Comparison of coreset performance on ijcnn1 dataset where size is the percentage of training data.

Size (%)	Method	F1 score	ROC	Accuracy	Time (seconds)
1	Full	0.46836	0.93313	0.92462	0.18
1	RCA	0.47105	0.89755	0.92229	0.49
1	ACvS	0.47105	0.89755	0.92229	0.03
1	RSDF	0.49022	0.90092	0.92261	0.03
1	URS	0.00123	0.81114	0.90412	0.01
3	Full	0.46836	0.93313	0.92462	0.19
3	RCA	0.47487	0.92229	0.92391	0.45
3	ACvS	0.47487	0.92229	0.92391	0.04
3	RSDF	0.44788	0.92085	0.92305	0.03
3	URS	0.08715	0.89298	0.90677	0.02
6	Full	0.46836	0.93313	0.92462	0.19
6	RCA	0.47861	0.92658	0.92456	0.46
6	ACvS	0.47861	0.92658	0.92456	0.04
6	RSDF	0.47063	0.92630	0.92477	0.04
6	URS	0.25508	0.91849	0.91363	0.02

Table 6.6: Comparison of coreset performance on the a9a data-set where size is the percentage of training data.

Size (%)	Method	F1 score	ROC	Accuracy	Time (seconds)
1	Full	0.65352	0.90178	85001	6.37423
1	Coreset	0.54603	0.87389	82500	0.42522
1	ACvS	0.54603	0.87389	82500	0.11467
1	RDSF	0.57807	0.87766	83000	0.11635
1	URS	0.58454	0.86766	82533	0.05234
3	Full	0.65352	0.90178	85001	7.37372
3	Coreset	0.61545	0.89065	84003	0.56512
3	ACvS	0.61545	0.89065	84003	0.23552
3	RDSF	0.61994	0.88761	83786	0.23287
3	URS	0.61978	0.88794	83889	0.13645
6	Full	0.65352	0.90178	85001	6.62791
6	Coreset	0.62880	0.89460	84228	0.75048
6	ACvS	0.62880	0.89460	84228	0.42837
6	RDSF	0.63791	0.89354	84257	0.42987
6	URS	0.63717	0.89554	84480	0.26902

It is not knowledge, but the act of learning, not possession but the act of getting there, which grants the greatest enjoyment. When I have clarified and exhausted a subject, then I turn away from it, in order to go into darkness again; the never-satisfied man is so strange if he has completed a structure, then it is not in order to dwell in it peacefully, but in order to begin another. I imagine the world conqueror must feel thus, who, after one kingdom is scarcely conquered, stretches out his arms for others ...

— Carl Friedrich Gauss
Letter to Bolyai, 1808.

7 Conclusion

This chapter summarises the findings of this investigation and places the contributions herein within the frames of the current status quo in machine learning. Viable future work on the line of research conisdered in the thesis will also be presented and discussed, along with possible applications.

7.1 Thesis Summary

The contributions presented in this research target the computational overhead imposed by conformal predictors and Venn-Abers predictors to traditional scoring classifiers. This issue, is partially addressed by considering the inductive version of these methods. The protocols defined in this thesis, C-ICP and IVAP-WEB, however, can further improve this computational gain, in some instances, by orders of magnitude. There are at least 3 interesting advantages of these propossed coreset-based approaches:

- C-ICP and IVAP-WEB do not add any additional assumptions on the data apart from those already required by CP and IVAP *e.g.* the IID property should hold for the input data. Furthermore, our methods are application-agnostic and hence they could be applied in any application where CP and IVAP can be applied.

- equipped with the aggregation properties of coresets, discussed briefly in Section 3.8, C-ICP and IVAP-WEB can be extended to different computational settings such as the streaming setting or the distributed setting. Furthermore, considering that CP was defined as a purely online method, which resembles the streaming setting very closely, this extension will be a natural fit for our methods.

- CP and VAP are simple in principle and they have a modular nature in the sense that they encourage a clear separation between the data summarisation part and the learning process. Thus, the learning process does not need to get informed about the existence of a compression step. This makes the task of exploring different coreset algorithms straighforward.

During this research effort, it was also observed that the construction of coresets could be very expensive, in terms of computational power, for the problem of logistic regression in the optimisation setting. This brings to light an interesting observation on the framework of coresets: the fact that there exists a theoretically proved procedure for finding a coreset for a specific problem, does not necessarily imply that there exists an efficient algorithm that implements the procedure. This work's response to such dilemma, for the specific problem of LR and its coreset construction proposed by Huggins *et al.* ([78]), is to accelerate the intrinsic clustering step in the coreset construction. This is achieved by the ACvS procedure, which exploits the fact that, for the problem of clustering, taking a small uniform sample of the data gives, in expectation, the same result. ACvS, overcoming the bottleneck in the coreset construction algorithm, allows the compression of input data into a coreset to happen substantially faster; at the same time, the procedure retains *exactly* the accuracy of the coreset algorithm. This last observation points to the conclusion that, for constructing a coreset for LR, a *rough* approximation to the clustering of the input data is sufficient for obtaining a coreset.

Based on the AcvS method, the RDSF protocol was designed: a meta framework that uses sensitivity-based coreset algorithms to produce high-quality summaries of

data that are complemented with a regression model. Thus, each RDSF summary is equipped with a regressor that retains valuable sensitivity information about the input points that could be used to assign, in constant time, importance scores to new data. RDSF hence uses the time saved by ACvS to apply machine learning to learn and predict sensitivities; and as observed in the numerical experiments presented, the sensitivities can be very accurately represented by a simple linear model. This sheds light on the mathematical nature of sensitivity scores and, interestingly, RDSF opens the research path for studying sensitivities as measures of independent interest, even outside of the framework of coresets.

Finally, ACvS and RDSF are independent of the ML algorithms of LR and SVM. This means that they could be applied to any problem that allows for a clustering-based coreset construction. In addition, this also means that, in principle, ACvS and RDSF can be used alongside our C-ICP and IVAP-WEB approaches, to provide them with even further acceleration.

7.2 Related Machine Learning Approaches

It is worth mentioning, to complement the results exposition of the thesis, some machine learning tools that can closely compare to the main ones used in this work.

7.2.1 Related to Coresets

Chapter 3 presented coresets as a young approach to data compression in machine learning. There are some alternative approaches to coresets that are worth mentioning in this section. The first one is **P**rincipal **C**omponent **A**nalysis (PCA) ([9], Ch. 14.5), which is an instance of an unsupervised machine learning problem that aims to project the dimensions of a data-set onto the so called *principal components*. The advantage of PCA is that it is a well-studied method in the ML community and it is widely available in modern software packages *e.g.* Python's *Scikit-learn* [1], R's *factoextra* [2]. On the other hand, PCA cannot offer a guarantee

[1] https://github.com/scikit-learn/scikit-learn - last accessed 20/05/2022
[2] https://cran.r-project.org/web/packages/factoextra/ - last accessed 20/05/2022

on the quality of the obtained projections and thus, the performance obtained on the reduced data-set could vary arbitrarily with respect to that obtained on the original data-set (see [125] and [126], and references therein, for some examples of this). It is important to mention, however, that computing PCA itself can be approximated with a coreset *i.e.* instead of computing the PCA of a large data-set one could compute the PCA for its coreset ([67]), and thus, PCA can also benefit from the approximations guaranteed by coresets.

Another technique that is widely used for representing data is ***t*-*d*istributed *s*tochastic *n*eighbor *e*mbedding** (t-SNE), propossed by Van der Maaten*et al.* ([127]). This method projects the original input points to low-dimensional *embeddings* and it has been successfully used to *understand* high dimensional data ([128]). Furthermore, it is even possible to combine t-SNE with some kernel methods to improve data representation in some applications ([129]). However, interpreting t-SNE representations seems to be more an art than a science (see [130] and [131]), and thus considerable work has to be put in *tuning* t-SNE adequately for representing input data. Also, as PCA, t-SNE does not provide any theoretical guarantees for the *quality* of the obtained representations. Therefore, its use is mainly restricted to data visualisation.

Finally, an important method for representing data that comes from theoretical computer science is that of *Property Testing* (PT) ([132]), which defines some formal tests in order to *decide* whether some well-defined properties hold in the given data-set. Thus, each test can be seen as a *decision problem*. The main difference between PT and coresets relies in their assumption on the availability of data. Coreset algorithms need to inspect every input point in the data-set *at least once* in order to define their relative importance with respect to some optimisation problem. PT methods, on the other hand, only look at *a portion* of the data, and they make an *approximate* decision, via the tests they apply, regarding some properties of interest. PT hence has the freedom of allowing *fast sub-linear time* algorithms, while coresets need *at least* linear time algorithms. The tests PT methods use, however, are challenging to design, and coreset algorithms are easier to implement.

Furthermore, PT does not perform any data compression; it can only *approximately* tell the user whether some structural properties hold in the data-set ([133]).

7.2.2 Related to Conformal Prediction

Conformal Prediction is a set of tools designed as a response to the lack of practical approaches for *measuring uncertainty* in ML. As stated in Chapter 4, CP sits at a level of abstraction which is *higher* than *traditional* machine learning methods *i.e.* ML algorithms are used within CP to obtain *significance levels* for the predictions made. Alternatives to CP, then, can be considered from two perspectives:

- *a theoretical alternative* is to just rely on the error bounds given by standard *Statistical Learning Theory* ([105]). More specifically, one can obtain a **P**rovably **C**orrect **A**pproximation (PAC) ([2], Ch. 3) of the performance of the machine learning algorithms of interest.

- *a practical alternative* is to simply rely on the traditional *hold-out estimate* for obtaining *some* confidence on the performance of the machine learning model trained over the training set one possesses ([13], p. 4) *i.e.* one splits all the available data into a training set and a test set, and the performance of the learning model, constructed over the training set, over the test set will indicate the confidence that one can expect from the model.

The downside of PAC theory is that error bounds given are usually *too loose* to be useful in practice ([104]). One important advantages of CP is that the error bounds obtained are *tight* and hence allow the user to reveal *informative* properties of the data-set she possesses *e.g.* whether the data is truly randomly generated. Therefore, CP can answer a more intricate question when compared to PAC: it tells us *how* approximate and *how* correct the predictions are.

Regarding the practical aspect, CP can give the ML practitioner significance *levels* for the predicted values even*before* looking at the *ground-truth* label of the test object in question. Furthermore, these significance levels are *guaranteed* to hold as long as we are learning over IID samples. This formal measure of *trust*

in the predictions is an advantage one cannot get from the traditional hold-out estimate approach.

7.2.3 Related to Venn-Abers Prediction

The most straightforward alternative to VAP is *Platt's scaling*, which is a simple calibrator that can prove useful *when* one does not worry about obtaining any measure of proximity to the *ground-truth* probabilities for observing the correct labels of the classification objects at hand. In fact, Chapter 4 gives an example of this kind of situations. Specifically, Section 4.2.7 highlights the fact that *high probability calibration* is not necessary when one is interested in computing non-conformity scores using the MEF approach in conformal prediction. This is because CP only uses non-conformity scores α_i to compare whether they are bigger, smaller or equal with respect to each other *i.e.* it does not care how close they are to their ground-truth probabilities (see Equation (4.4)). A somewhat similar approach to Platt's scaling is that of *isotonic regression*, which is also widely used as calibrator (see the study in ([134]) for details). However, just like Platt's scaling, isotonic regression cannot tell us *how far* the obtained probabilities are from the true probabilities. *That* piece of information on the proximity to the true probabilities is what makes VAP a quite unique and powerful calibrator. Unlike the just-mentioned alternatives, VAP outputs multi-probabilistic results, which are in fact *bounds*; and the VAP user is *guaranteed* that the ground-truth probability of observing the positive label for a given object is contained within those bounds. The comparison among Platt's Scaling, isotonic regression and VAP is well-documented in [16] and references therein, and also in the work of Vovk *et al.* ([14]).

7.3 Future Work

This investigation leaves the door open for many follow-up research works. A subset of them are formulated below in the form of research questions.

- **What is the performance of coreset-based approaches when applied to other members of the RML family?** - This is a natural question to ask after observing the results of C-ICP and IVAP-WEB. Even though we have covered the two fundamental methods of Conformal Prediction and Venn-Abers Prediction, an interesting research to undertake is that of testing this idea with frameworks such as Venn Predictors ([13], Ch. 6, p. 158), Mondrian-conformal Predictors ([13], Ch. 4, p. 114) and Cross-conformal Prediction ([135]).

- **Can the concept of non-conformity measure from Conformal Prediction and the concept of sensitivity from Coresets be reconciled?** - Based on our observations, despite being proposed for entirely different problems at very different communities, it seems that developing a connection between these two topics can be an interesting research project. For example, if we manage to obtain a good approach for computing sensitivities bringing the notion of non-conformity from CP, we could produce an interesting method for compressing input data.

- **What would the performance of the RDSF framework be in the streaming setting?** - Given that RDSF allows one to retain a regression model based on the sensitivities of input points, an interesting question to ask is whether RDSF will help us decide quicker, in a streaming setting, which incoming points are the most important ones. With this knowledge, we can develop a strategy to retain data that favours points with high sensitivity.

- **Can labelling information be incorporated in the computation of coresets to, at least empirically, make them robust to pathological input data?** - The obtained results showed that imbalance can severely hurt the performance of machine learning algorithms, and thus handling this situation is crucial. Since coreset-construction algorithms available for classification usually rely on primitive coreset algorithms for unsupervised problems, label information is not taken into account in the compression

process. Thus, coresets are vulnerable to imbalance data in supervised settings. Making a balanced compression of imbalanced input data could hence potentially improve the performance of classifiers.

Appendices

The oldest and strongest emotion of mankind is fear, and the oldest and strongest kind of fear is fear of the unknown.

— H.P. Lovecraft

A

Complementary Results on the C-ICP Protocol

Contents

This appendix offers some further results obtained with the new learning protocol of Coreset-based Inductive Conformal Prediction (C-ICP). The results cover the same list of data-sets used in Chapter 4, and described in Chapter 2. Section A.1 shows results of validity and efficiency of ICP and C-ICP for different significance levels for the SVM problem. Section A.2 tables showing results for some specific significance levels.

Finally, Section A.3 shows results of instantiating C-ICP with a coreset algorithm as an heuristic. To remind the reader, a coreset construction algorithm can be used for any problem of interest at the expense of losing the theoretical guarantees they offer as they only provide guarantees for the problem they were designed. Therefore, we will use AVM, originally designed for SVM, for summarising data

for LR within the context of C-ICP. Similarly, we will instantiate C-ICP for the problem of SVM using the RCA algorithm, originally designed for the problem of LR. These results show hence the modular nature of C-ICP, and also they shed light on using coresets as general-purpose compression technique.

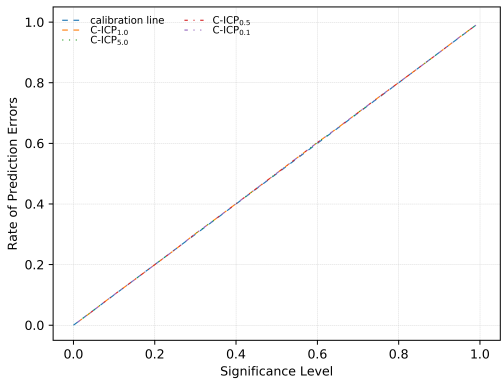
A.1 C-ICP and ICP for SVM

Figures A.1, A.2, A.3, A.4, A.5 and A.6 complement the results shown in Section 4.4.4 with those obtained for the SVM problem, for all the data-sets in Table 2.1.

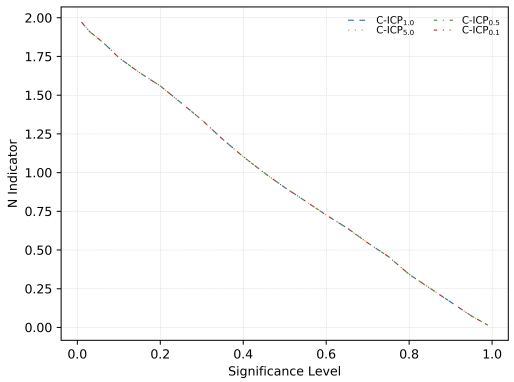
A.2 Validity and Efficiency Tables for C-ICP, ICP and URS

It is useful to consider the performance of training ICP over a random uniform sample (URS) of the proper training set. As we have mentioned on Section 3, the most computationally efficient approach for reducing the size of data-set is to simply assume equal probability for all input points and take as many points as necessary uniformly. Even though this approach cannot provide us with small summaries that consistently approximate sets of data with strong theoretical guarantees, it is still useful, specially for assessing the practical performance of coresets. Hence, we are also interested in showing how the results obtained from URS compares to those obtained from coresets, within the framework of CP. This information is presented in the tables from this section.

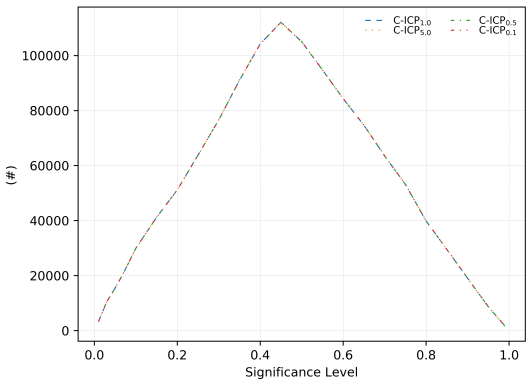
Table A.1 and Table A.2 present the validity and efficiency results for the data-sets in the Desirable group. Each table present the measures as rows and the methods as columns. Furthermore, each row shows the result of the methods for both LR and SVM. Underscores signify that measures could not be taken for the specific method due to a violation of our time budget. When the nature of the metric allows it, we present in the table both the rates and the non-normalised numbers, signified with R and #, respectively, in order to make comparisons easier.



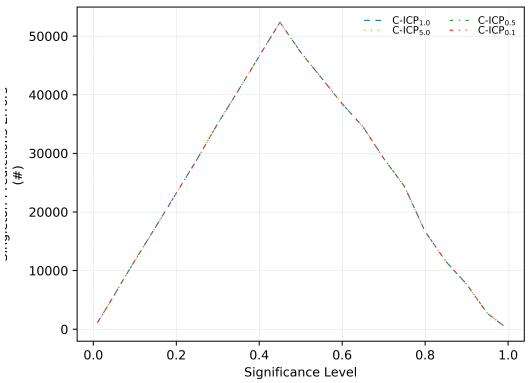
(a) Errors



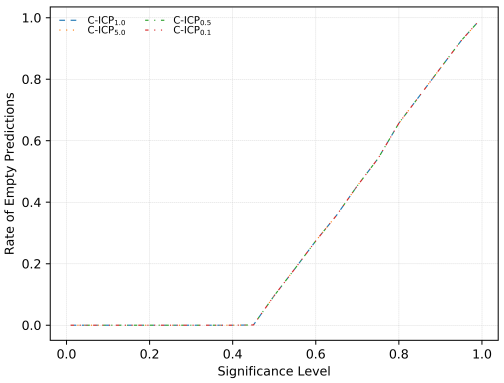
(b) Average Size of Prediction Sets (N Indicator)



(c) Singleton Predictions



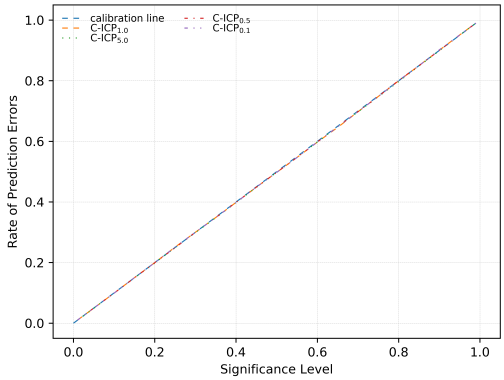
(d) Singleton Errors



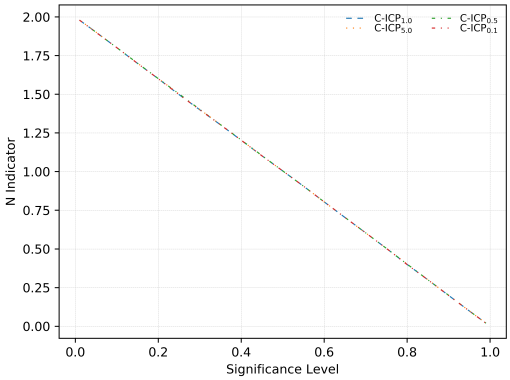
(e) Empty Predictions

Figure A.1: Validity and efficiency measures obtained by C-ICP and ICP at different significance levels on the covertype data-set for the SVM problem.

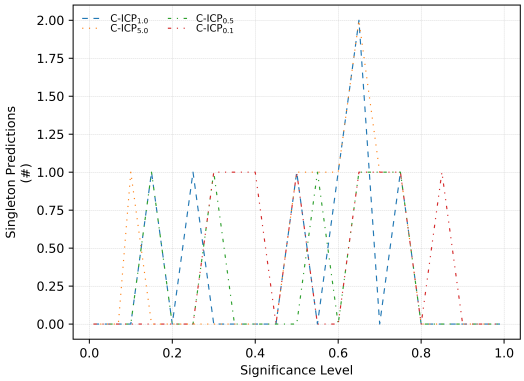
Fixing our focus to the results on covertype for LR on Table A.1, we can start by mentioning a very important observation: to our surprise, C-ICP shows the



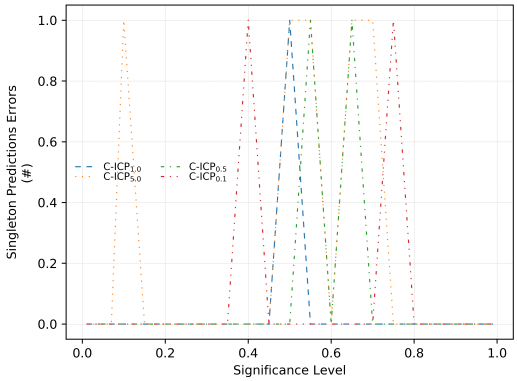
(a) Errors



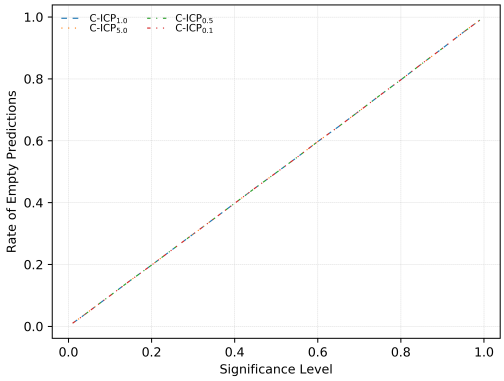
(b) Average Size of Prediction Sets (N Indicator)



(c) Singleton Predictions



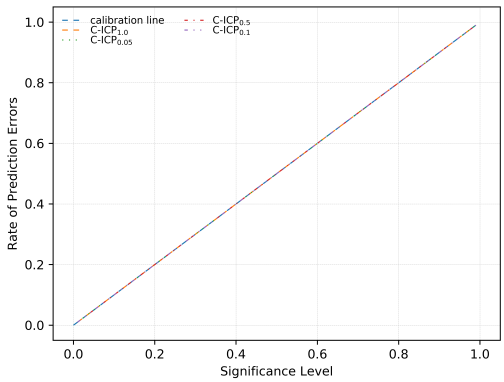
(d) Singleton Errors



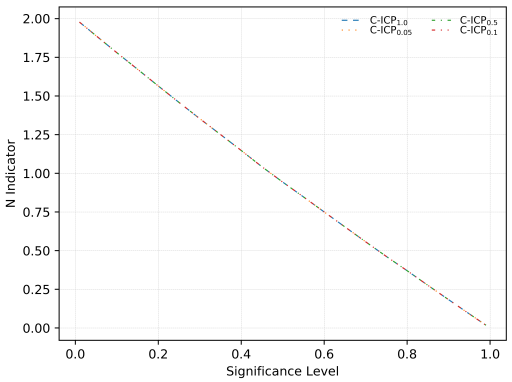
(e) Empty Predictions

Figure A.2: Validity and efficiency measures obtained by C-ICP and ICP at different significance levels on the webspam data-set for the SVM problem.

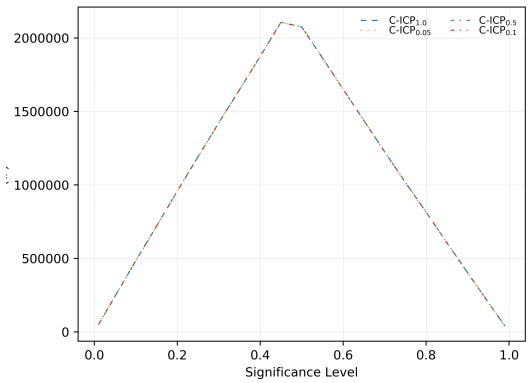
best performance of the three methods. That is, by using only a coreset of 1% of the full input data, C-ICP obtained not only better performance than URS, but



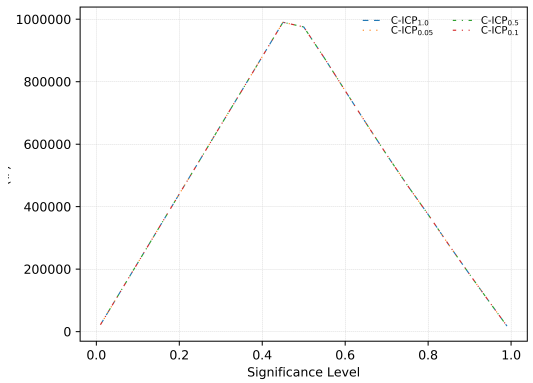
(a) Errors



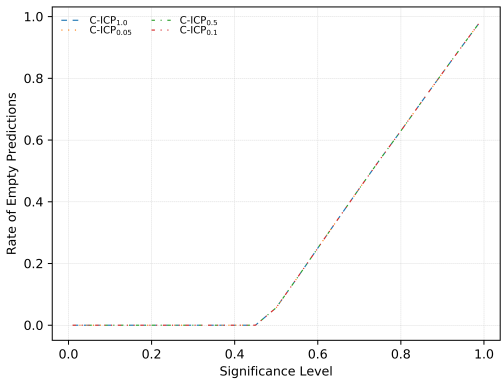
(b) Average Size of Prediction Sets (N Indicator)



(c) Singleton Predictions



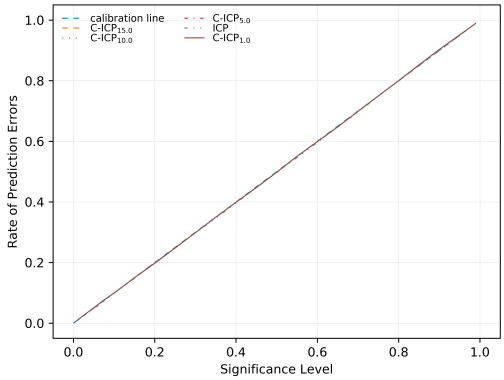
(d) Singleton Errors



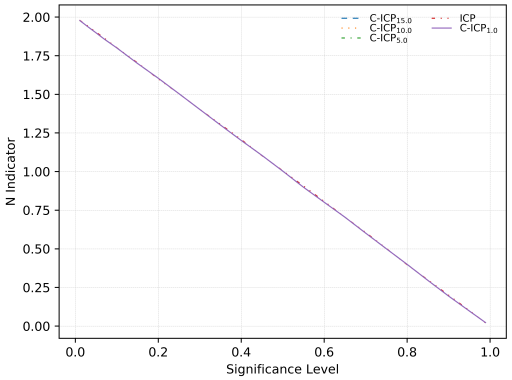
(e) Empty Predictions

Figure A.3: Validity and efficiency measures obtained by C-ICP and ICP at different significance levels on the higgs data-set for the SVM problem.

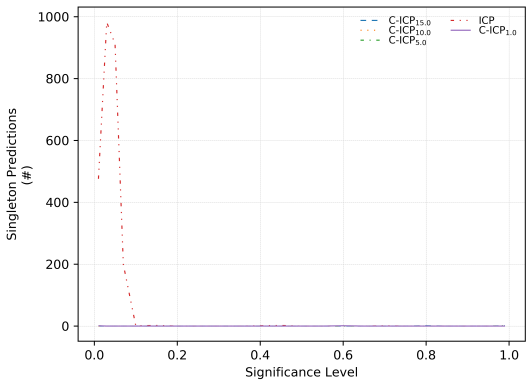
also better performance than standalone ICP. This can be seen by inspecting (i) the number of singleton predictions ($\mathcal{S}$); (ii) the number of erroneous singleton



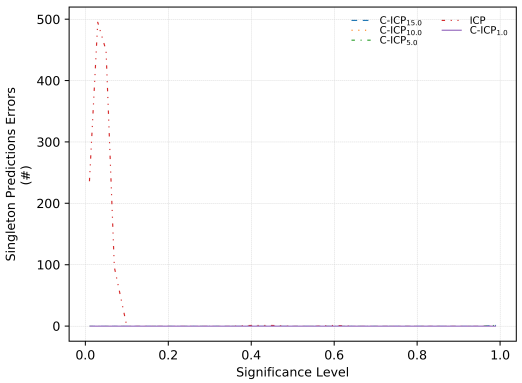
(a) Errors



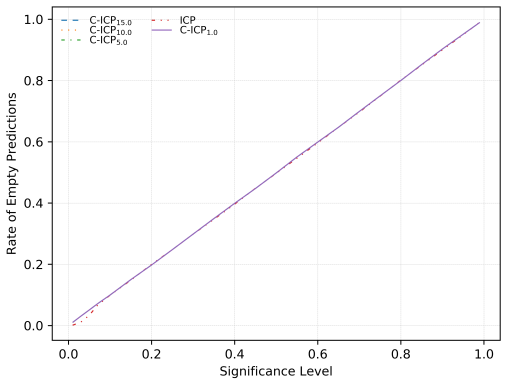
(b) Average Size of Prediction Sets (N Indicator)



(c) Singleton Predictions



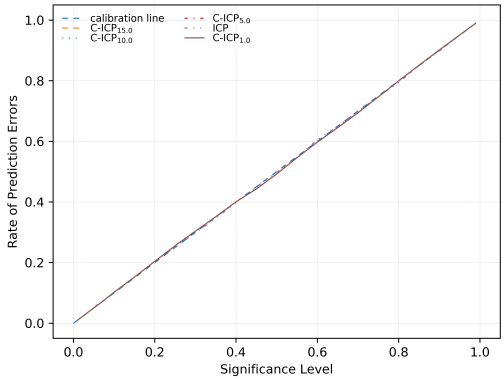
(d) Singleton Errors



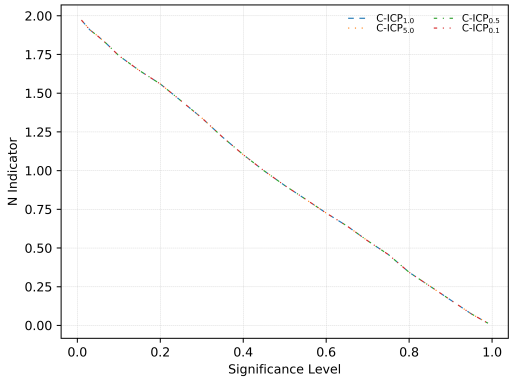
(e) Empty Predictions

Figure A.4: Validity and efficiency measures obtained by C-ICP and ICP at different significance levels on the ijcnn1 data-set for the SVM problem.

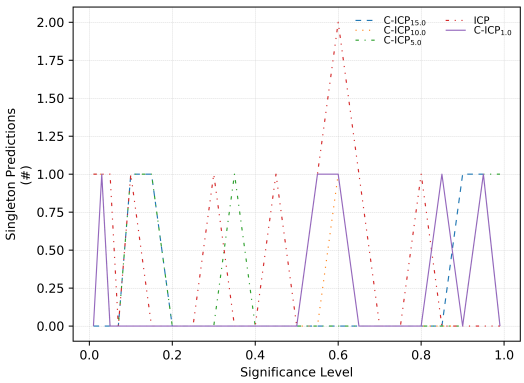
predictions ($\mathcal{E}$ ($\mathcal{S}$)). By comparing the performance of C-ICP against that of ICP, we see that C-ICP makes more singleton predictions (104,365 for C-ICP, 104,274



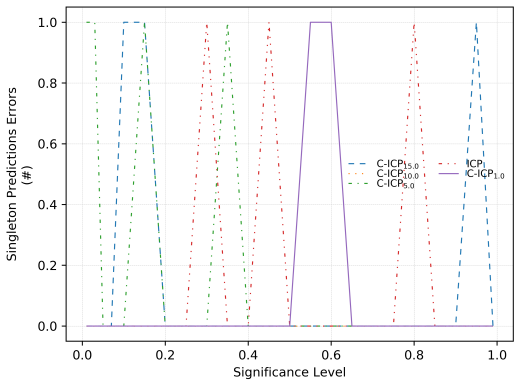
(a) Errors



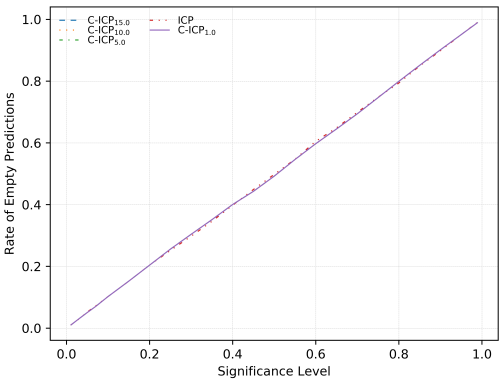
(b) Average Size of Prediction Sets (N Indicator)



(c) Singleton Predictions



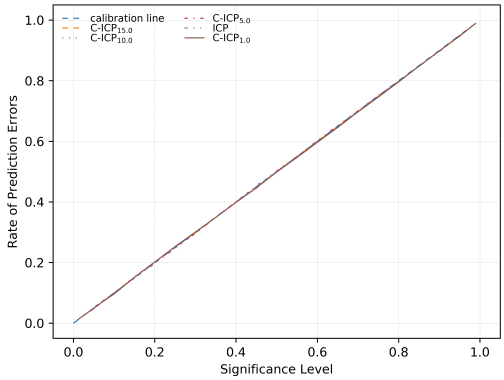
(d) Singleton Errors



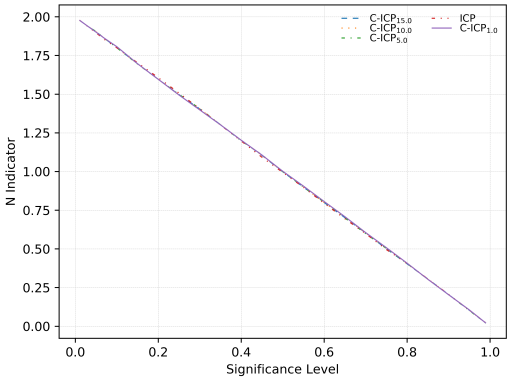
(e) Empty Predictions

Figure A.5: Validity and efficiency measures obtained by C-ICP and ICP at different significance levels on the a9a data-set for the SVM problem.

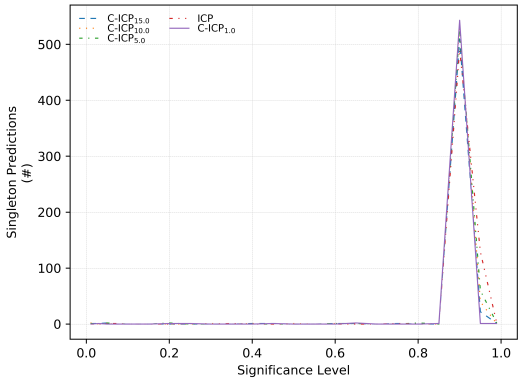
for ICP), and also, a smaller number of those predictions are mistaken (23,091 for C-ICP, 34,809 for ICP). Furthermore, C-ICP returns less empty predictions



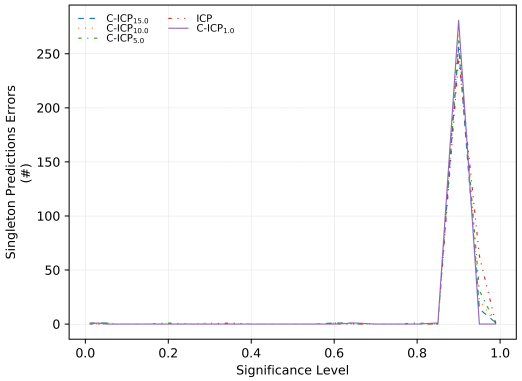
(a) Errors



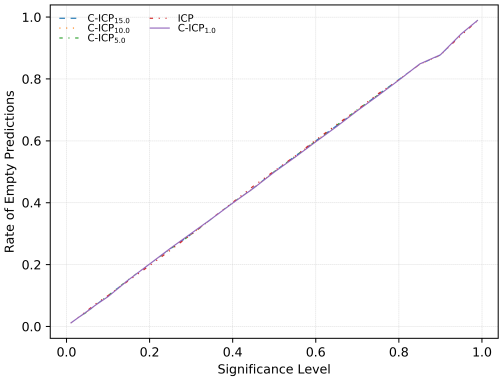
(b) Average Size of Prediction Sets (N Indicator)



(c) Singleton Predictions



(d) Singleton Errors



(e) Empty Predictions

Figure A.6: Validity and efficiency measures obtained by C-ICP and ICP at different significance levels on the w8a data-set for the SVM problem.

when compared to ICP. We find this result stimulating as they can impulse new research lines into directions where coresets could be used to boost accuracy. This

Table A.1: Validity and efficiency measures for covertype. $\boldsymbol{\mathcal{E}}$ signifies errors, in the CP sense; $\boldsymbol{\mathcal{S}}$ and $\boldsymbol{\mathcal{E}}$ ($\boldsymbol{\mathcal{S}}$) represent singleton predictions and errors in singleton predictions. **E** is empty predictions, **N** is the average size of the prediction sets obtained and $\boldsymbol{S}$ is the average sum of p-values. #|R means number of predictions | rate of predictions, respectively.

covertype				$\boldsymbol{\epsilon} = 0.3$
Measure	**Problem**	**ICP**	**C-ICP (1%)**	**URS-ICP (1%)**
$\boldsymbol{\mathcal{E}}$: #|R	LR	34,886 | 0.3002	34,929 | 0.3005	34,877 | 0.3001
	SVM	-	35,092 | 0.3019	34,809 | 0.2996
$\boldsymbol{\mathcal{S}}$: #|R	LR	104,274 | 0.8973	104,365 | 0.8981	113,325 | 0.9752
	SVM	-	107,368 | 0.924	76,623 | 0.6593
$\boldsymbol{\mathcal{E}}$ ($\boldsymbol{\mathcal{S}}$): #|R	LR	23,131 | 0.222133	23,091 | 0.222125	31,999 | 0.2824
	SVM	-	34,809 | 0.3258	35,092 | 0.4613
E: #|R	LR	11,929 | 0.1026	11,838 | 0.1018	2,878 | 0.0248
	SVM	-	0 | 0	0 | 0
N	LR	0.8973	0.8981	0.9752
	SVM	-	1.0760	1.3406
$\boldsymbol{S}$	LR	**0.3367**	0.3370	0.3596
	SVM	-	0.3946	0.4748

phenomenon seem to contradict the norm in ML, as generally more data means better predictive accuracy. However, one possible explanation we can offer is that coresets, by definition, tend to retain input points that contribute more toward the objective function at hand. Thus, in this particular case, it could be the case that coresets get rid of some noise present in the input data. For higgs (Table A.2), we can only compare C-ICP against URS as ICP is too computationally expensive for our time budget. The patterns seen between C-ICP and URS are similar for both covertype and higgs: for both LR and SVM, the URS method seem to be dangerously overconfident when making predictions; this is specially true for SVM on the coverype data-set, where the predictions by URS are 65.93% singleton, from which 46.13% are erroneous. C-ICP, on the other hand, makes predictions from which 92.4% are singleton, and only 32.58% of those predictions are mistaken. For LR on covertype, URS generates a larger number of singleton predictions (97.52%)

Table A.2: Validity and efficiency measures for webspam. $\boldsymbol{\mathcal{E}}$ signifies errors, in the CP sense; $\boldsymbol{\mathcal{S}}$ and $\boldsymbol{\mathcal{E}}$ ($\boldsymbol{\mathcal{S}}$) represent singleton predictions and errors in singleton predictions. **E** is empty predictions, **N** is the average size of the prediction sets obtained and $\boldsymbol{S}$ is the average sum of p-values. #|R means number of predictions | rate of predictions, respectively.

higgs			$\boldsymbol{\epsilon} = 0.35$	
Measure	**Problem**	**ICP**	**C-ICP**	**URS-ICP (1%)**
$\boldsymbol{\mathcal{E}}$: #|R	LR	-	771,020 | 0.3504	771,398 | 0.3506
	SVM	-	769,840 | 0.3499	770,930 | 0.3504
$\boldsymbol{\mathcal{S}}$: #|R	LR	-	2,170,041 | 0.9866	215,0537 | 0.9775
	SVM	-	2,041,537 | 0.928	1,648,178 | 0.7491
$\boldsymbol{\mathcal{E}}$ ($\boldsymbol{\mathcal{S}}$): #|R	LR	-	771,156 | 0.3548	771,398 | 0.3585
	SVM	-	770,930 | 0.3777	769,840 | 0.4670
E: #|R	LR	-	0 | 0	0 | 0
	SVM	-	0 | 0	0 | 0
N	LR	-	1.0133	1.0225
	SVM	-	1.0720	1.2508
$\boldsymbol{S}$	LR	-	0.404	0.4062
	SVM	-	0.4219	0.4823

compared to C-ICP (89.81%); however, a larger portion of the URS predictions (28.24%) are mistaken, compared to the C-ICP predictions (22.21%). In this case, C-ICP is more conservative and thus produce more accurate predictions. Finally, the performance of URS is much closer to that of C-ICP for higgs for the LR problem; still, C-ICP does give preferable solutions when analysing the metrics values.

Tables A.3, A.4, A.5 and A.6 show results for the webspam, ijcnn1, a9a and w8a, respectively. First, it is important to notice that these data-set suffer from different degrees of imbalance (see the information in Table 2.1), and this seem to impact the performance of conformal predictors in general. The main pattern observed on the data-sets in the Pathological group is that the three methods seem to either output prediction sets that are empty or full *i.e.* the two extremes of non-informative efficiency. When this is the case, the performances of standalone ICP, C-ICP and URS become indistinguishable. This means that the p-values for

Table A.3: Validity and efficiency measures for webspam. $\boldsymbol{\mathcal{E}}$ signifies errors, in the CP sense; $\boldsymbol{\mathcal{S}}$ and $\boldsymbol{\mathcal{E}}$ ($\boldsymbol{\mathcal{S}}$) represent singleton predictions and errors in singleton predictions. **E** is empty predictions, **N** is the average size of the prediction sets obtained and $\boldsymbol{S}$ is the average sum of p-values. #|R means number of predictions | rate of predictions, respectively.

webspam		$\boldsymbol{\epsilon} = 0.01$		
Measure	**Problem**	**ICP**	**C-ICP (1%)**	**URS-ICP (1%)**
$\boldsymbol{\mathcal{E}}$: #|R	LR	719 | 0.01027	715 | 0.01021	721 | 0.0103
	SVM	-	721 | 0.0103	705 | 0.010
$\boldsymbol{\mathcal{S}}$: #|R	LR	0 | 0	0 | 0	0 | 0
	SVM	-	0 | 0	0 | 0
$\boldsymbol{\mathcal{E}}$ ($\boldsymbol{\mathcal{S}}$): #|R	LR	0 | 0	0 | 0	0 | 0
	SVM	-	0 | 0	0 | 0
E: #|R	LR	719 | 0.01027	715 | 0.01021	721 | 0.0103
	SVM	-	721 | 0.0103	705 | 0.0101
N	LR	**1.9794**	1.9795	1.9794
	SVM	-	1.9794	1.9799
$\boldsymbol{S}$	LR	**0.5004**	0.5007	0.5011
	SVM	-	0.5006	0.5007

each prediction on this datasets are too close to each other that the methods either accept or reject both of them. We can interpret this as CP telling us that, from the data seen, it is not possible to find meaningful patterns to distinguish objects from different classes. Thus, from the information obtained in the Pathological cases, it is not possible to decide which method performs better. It might be tempting to assume that the results on the Pathological group will not help much in our analysis. However, these results very useful as CP is warning us about potential pathological patterns in the input data that might require closer inspection.

A.3 C-ICP with Coresets as Heuristics

In this section, we show the results of instantiating C-ICP with a coreset algorithm as a heuristic. Specifically, Section A.3.1 presents Figures A.7, A.8, A.9, A.10,

Table A.4: Validity and efficiency measures for ijcnn1. $\boldsymbol{\mathcal{E}}$ signifies errors, in the CP sense; $\boldsymbol{\mathcal{S}}$ and $\boldsymbol{\mathcal{E}}$ ($\boldsymbol{\mathcal{S}}$) represent singleton predictions and errors in singleton predictions. **E** is empty predictions, **N** is the average size of the prediction sets obtained and $\boldsymbol{S}$ is the average sum of p-values. #|R means number of predictions | rate of predictions, respectively.

ijcnn1		$\epsilon = 0.05$		
Measure	**Problem**	**ICP**	**C-ICP (1%)**	**URS-ICP (1%)**
$\boldsymbol{\mathcal{E}}$: #|R	LR	1,404 | 0.049543	1,403 | 0.049507	1,406 | 0.0496
	SVM	1,379 | 0.0486	1,444 | 0.0509	1371 | 0.0484
$\boldsymbol{\mathcal{S}}$: #|R	LR	0 | 0	1 | 3.5e-05	1 | 3.5e-05
	SVM	**912 | 0.0321**	0 | 0	0 | 0
$\boldsymbol{\mathcal{E}}$ ($\boldsymbol{\mathcal{S}}$): #|R	LR	0 | 0	0 | 0	0 | 0
	SVM	449 | 0.5021	0 | 0	0 | 0
E: #|R	LR	1,404 | 0.049543	1,403 | 0.049507	1,405 | 0.0496
	SVM	950 | 0.0335	1,444 | 0.0509	1371 | 0.0484
N	LR	**1.90091**	1.90098	1.9007
	SVM	1.9025	1.8980	1.9032
$\boldsymbol{S}$	LR	0.50079	0.50072	0.5007
	SVM	0.5010	0.5000	0.5012

A.11 and A.12, which show that C-ICP can effectively use RCA to compress data for the SVM problem. Finally, Section A.3.2 shows the results of instantiating C-ICP with a coreset algorithm as a heuristic for the problem of LR. In this case, AVM is used to construct the coreset. Figures A.13, A.14, A.15, A.16, A.17 and A.18 show these results.

Table A.5: Validity and efficiency measures for a9a. $\boldsymbol{\mathcal{E}}$ signifies errors, in the CP sense; $\boldsymbol{\mathcal{S}}$ and $\boldsymbol{\mathcal{E}}$ ($\boldsymbol{\mathcal{S}}$) represent singleton predictions and errors in singleton predictions. **E** is empty predictions, **N** is the average size of the prediction sets obtained and $\boldsymbol{S}$ is the average sum of p-values. #|R means number of predictions | rate of predictions, respectively.

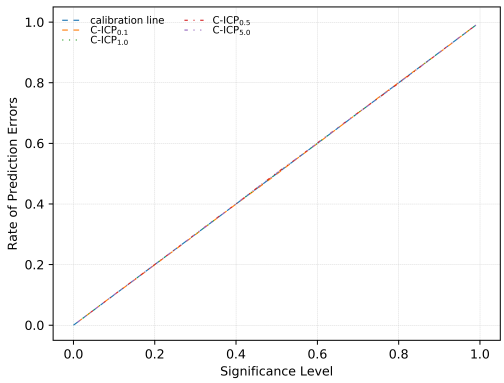
a9a		$\boldsymbol{\epsilon} = 0.05$		
Measure	**Problem**	**ICP**	**C-ICP (1%)**	**URS-ICP (1%)**
$\boldsymbol{\mathcal{E}}$: #|R	LR	497 | 0.0508	505 | 0.0516	463 | 0.0474
	SVM	509 | 0.0521	495 | 0.0506	470 | 0.0481
$\boldsymbol{\mathcal{S}}$: #|R	LR	0 | 0	2 | 0.2e-03	0 | 0
	SVM	**1 | 0.1e-03**	0 | 0	0 | 0
$\boldsymbol{\mathcal{E}}$ ($\boldsymbol{\mathcal{S}}$): #|R	LR	0 | 0	1 | 0.5	0 |0
	SVM	0 | 0	0 | 0	0 | 0
E: #|R	LR	497 | 0.0508	500 | 0.05011	463 | 0.0474
	SVM	509 | 0.0521	494 | 0.0505	469 | 0.048
N	LR	1.8982	1.8970	1.9052
	SVM	1.8956	1.8987	1.9039
$\boldsymbol{S}$	LR	0.5012	0.5011	0.4997
	SVM	0.5012	0.5025	0.5016

A.3.1 Plugging-in RCA to C-ICP for SVM

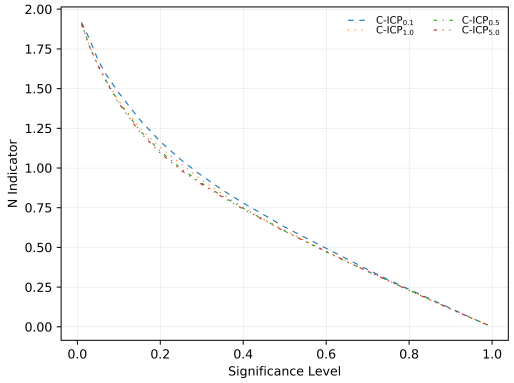
Table A.6: Validity and efficiency measures for w8a. $\boldsymbol{\mathcal{E}}$ signifies errors, in the CP sense; $\boldsymbol{\mathcal{S}}$ and $\boldsymbol{\mathcal{E}}$ ($\boldsymbol{\mathcal{S}}$) represent singleton predictions and errors in singleton predictions. **E** is empty predictions, **N** is the average size of the prediction sets obtained and $\boldsymbol{S}$ is the average sum of p-values. #|R means number of predictions | rate of predictions, respectively.

w8a		$\boldsymbol{\epsilon} = 0.9$		
Measure	**Problem**	**ICP**	**C-ICP**	**URS-ICP (1%)**
$\boldsymbol{\mathcal{E}}$: #|R	LR	11,648 | 0.9001	11,612 | 0.8973	11,638 | 0.8994
	SVM	11,599 | 0.8963	11,622 | 0.8981	11633 | 0.899
$\boldsymbol{\mathcal{S}}$: #|R	LR	533 | 0.0411	475 | 0.0367	482 | 0.0372
	SVM	491 | 0.0379	527 | 0.0407	0 | 0
$\boldsymbol{\mathcal{E}}$ ($\boldsymbol{\mathcal{S}}$): #|R	LR	262 | 0.4973	230 | 0.5030	251 |0.5207
	SVM	246 | 0.4927	262 | 0.4868	0 | 0
E: #|R	LR	497 | 0.0508	500 | 0.05011	11,448 | 0.8847
	SVM	11,362 | 0.8780	11,362 | 0.8780	11,633 | 0.899
N	LR	0.2019	0.2052	0.2018
	SVM	0.2059	0.2037	0.2020
$\boldsymbol{S}$	LR	0.5009	0.5004	0.4999
	SVM	0.5002	0.5010	0.4999

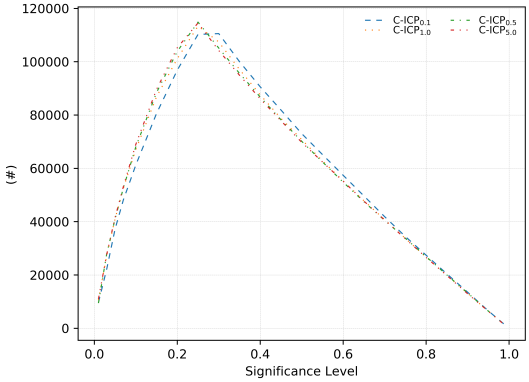
A.3.2 Plugging-in AVM to C-ICP for LR



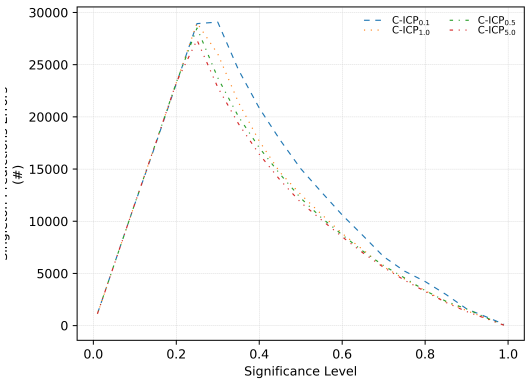
(a) Errors



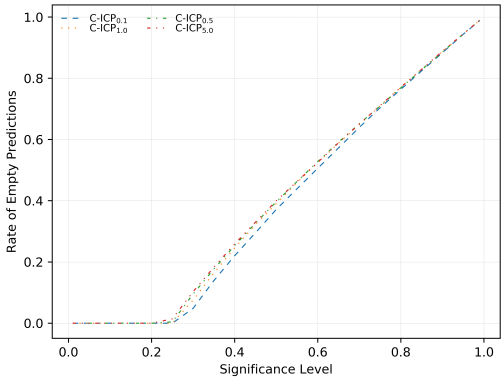
(b) Average Size of Prediction Sets (N Indicator)



(c) Singleton Predictions

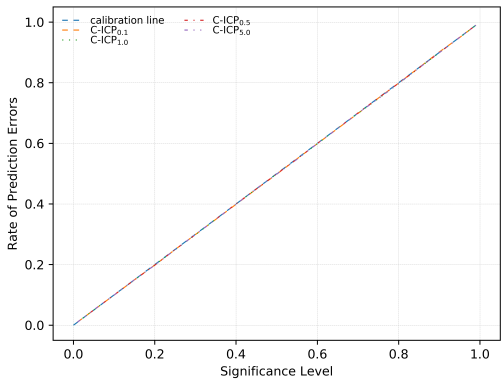


(d) Singleton Errors

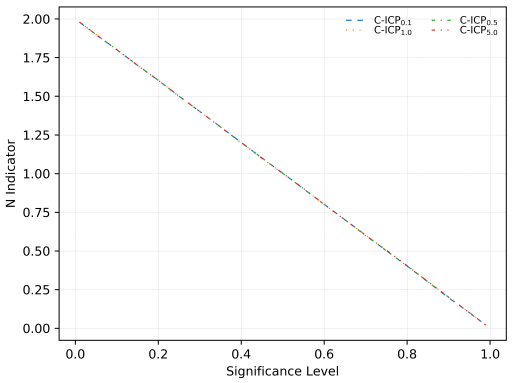


(e) Empty Predictions

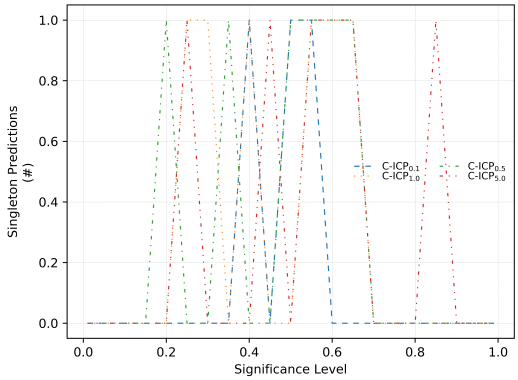
Figure A.7: Validity and efficiency measures obtained by C-ICP and ICP at different significance levels on the covertype data-set for the SVM problem. C-ICP uses RCA as a heuristic for compressing the data for SVM.



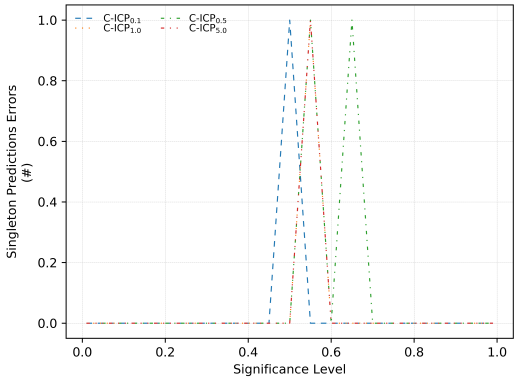
(a) Errors



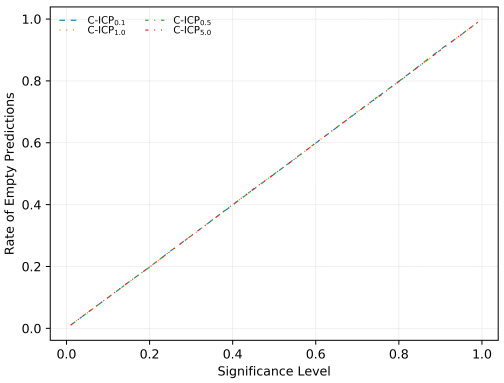
(b) Average Size of Prediction Sets (N Indicator)



(c) Singleton Predictions

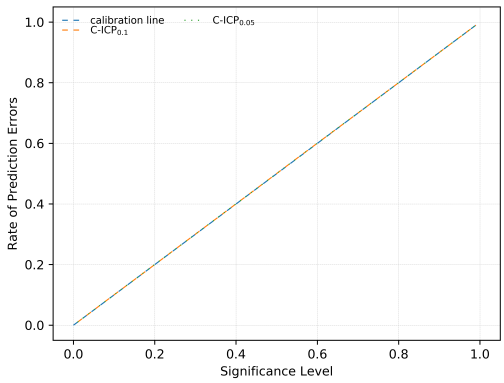


(d) Singleton Errors

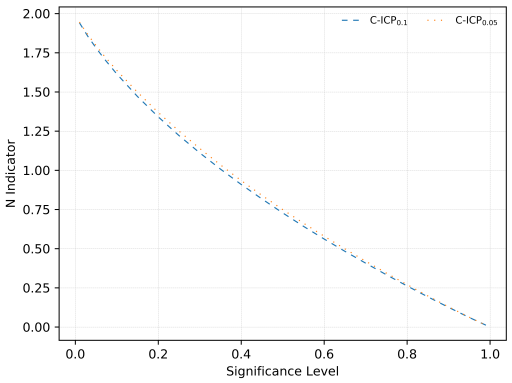


(e) Empty Predictions

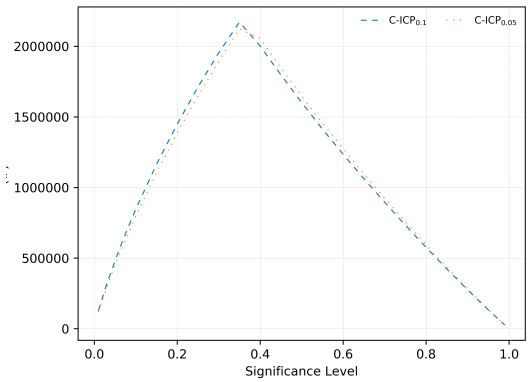
Figure A.8: Validity and efficiency measures obtained by C-ICP and ICP at different significance levels on the webspam data-set for the SVM problem. C-ICP uses RCA as a heuristic for compressing the data for SVM.



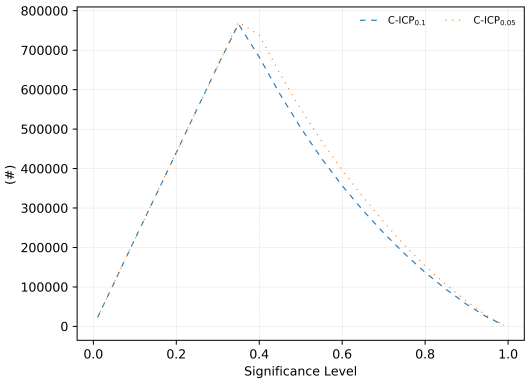
(a) Errors



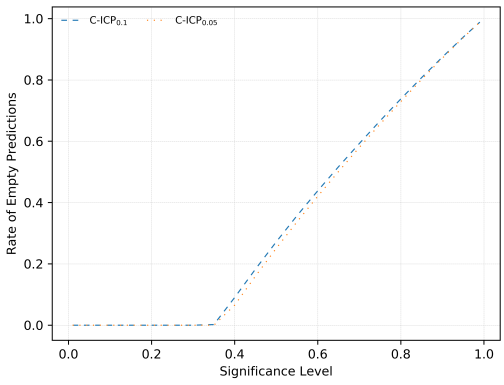
(b) Average Size of Prediction Sets (N Indicator)



(c) Singleton Predictions

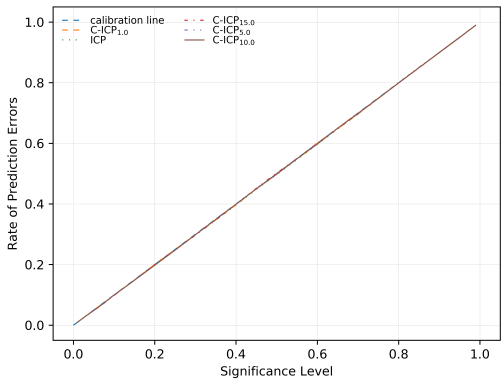


(d) Singleton Errors

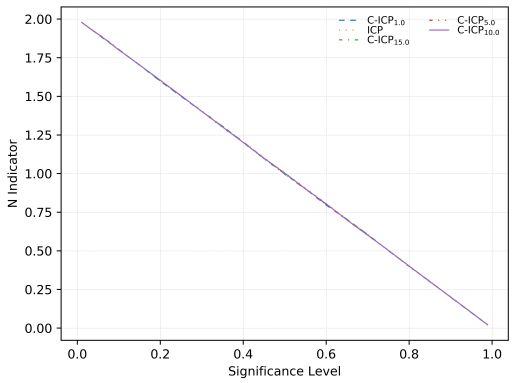


(e) Empty Predictions

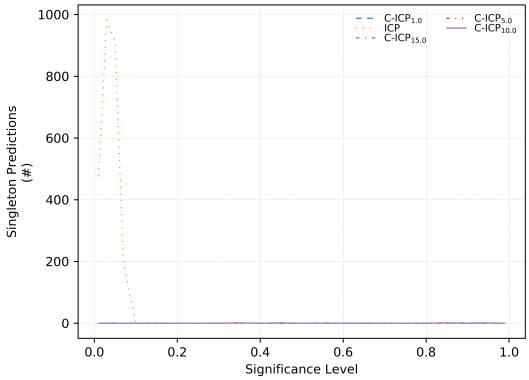
Figure A.9: Validity and efficiency measures obtained by C-ICP and ICP at different significance levels on the higgs data-set for the SVM problem. C-ICP uses RCA as a heuristic for compressing the data for SVM.



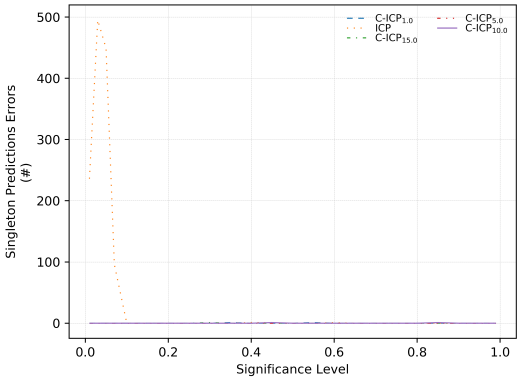
(a) Errors



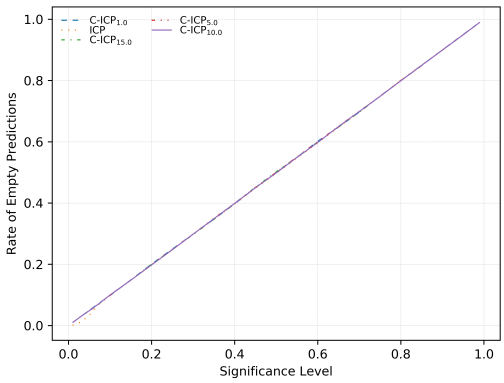
(b) Average Size of Prediction Sets (N Indicator)



(c) Singleton Predictions

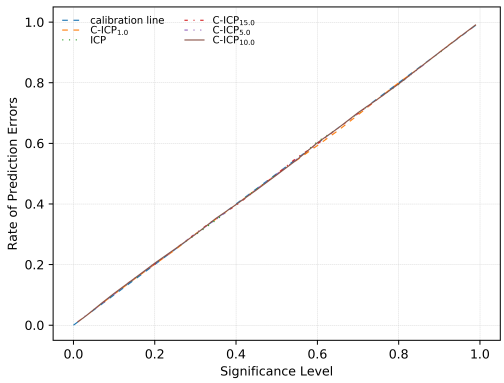


(d) Singleton Errors

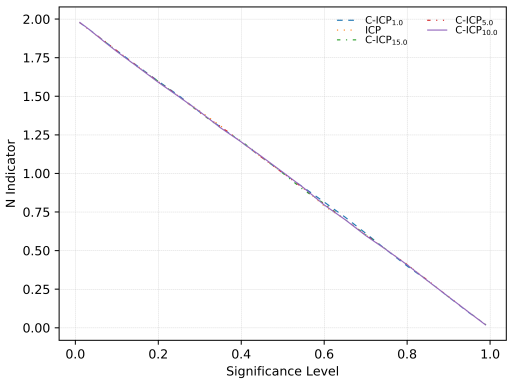


(e) Empty Predictions

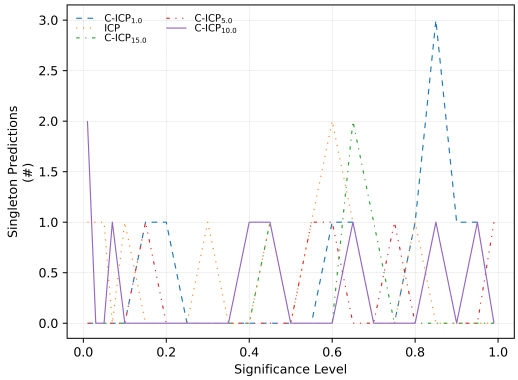
Figure A.10: Validity and efficiency measures obtained by C-ICP and ICP at different significance levels on the ijcnn1 data-set for the SVM problem. C-ICP uses RCA as a heuristic for compressing the data for SVM.



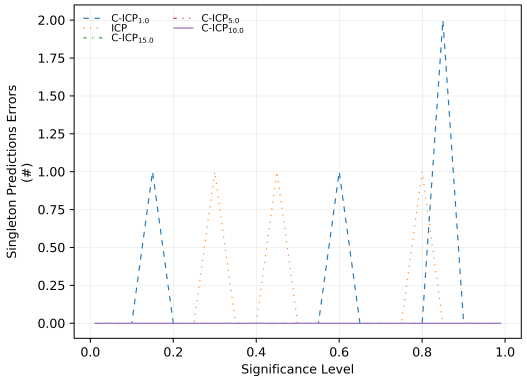
(a) Errors



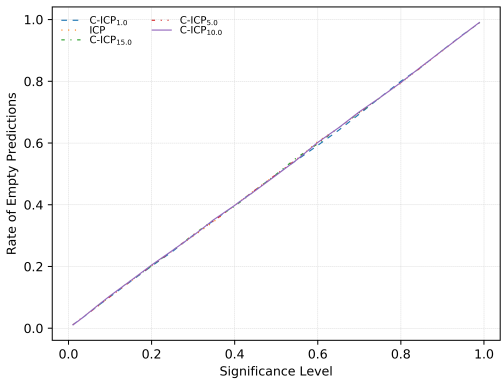
(b) Average Size of Prediction Sets (N Indicator)



(c) Singleton Predictions

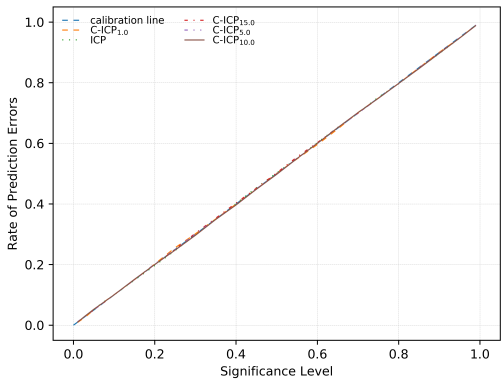


(d) Singleton Errors

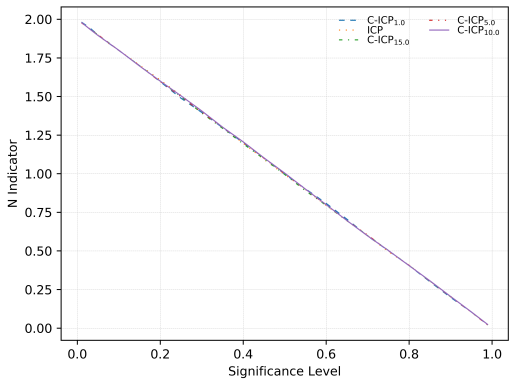


(e) Empty Predictions

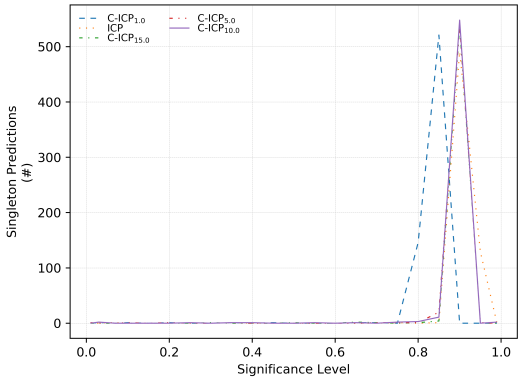
Figure A.11: Validity and efficiency measures obtained by C-ICP and ICP at different significance levels on the a9a data-set for the SVM problem. C-ICP uses RCA as a heuristic for compressing the data for SVM.



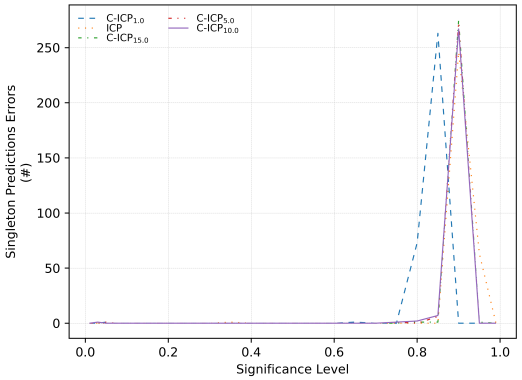
(a) Errors



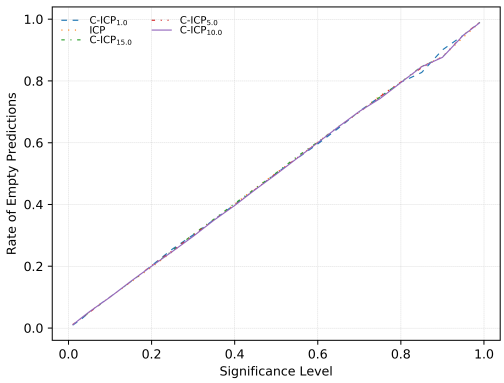
(b) Average Size of Prediction Sets (N Indicator)



(c) Singleton Predictions

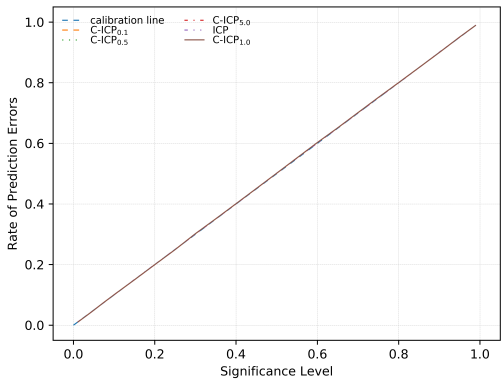


(d) Singleton Errors

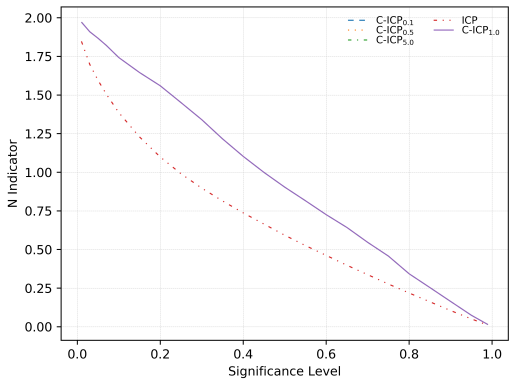


(e) Empty Predictions

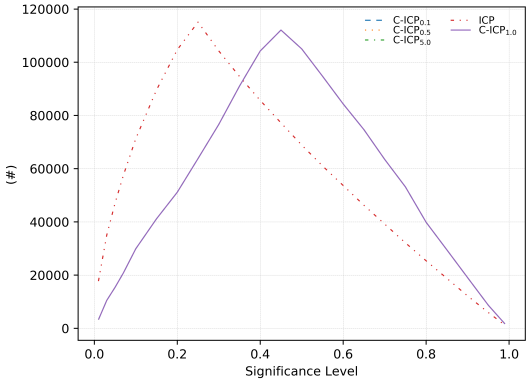
Figure A.12: Validity and efficiency measures obtained by C-ICP and ICP at different significance levels on the w8a data-set for the SVM problem. C-ICP uses RCA as a heuristic for compressing the data for SVM.



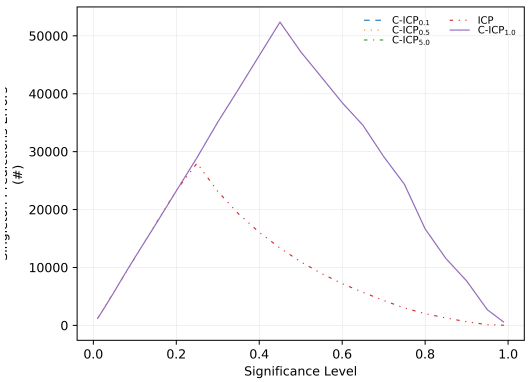
(a) Errors



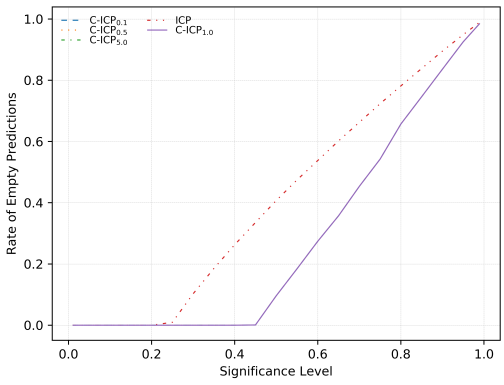
(b) Average Size of Prediction Sets (N Indicator)



(c) Singleton Predictions

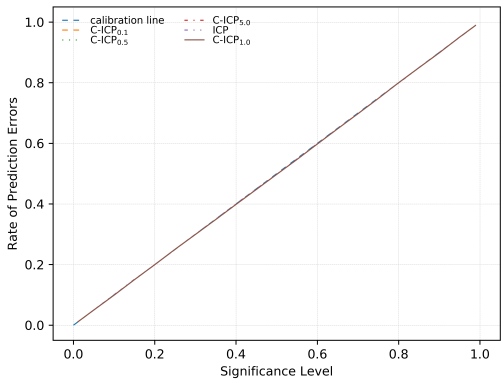


(d) Singleton Errors

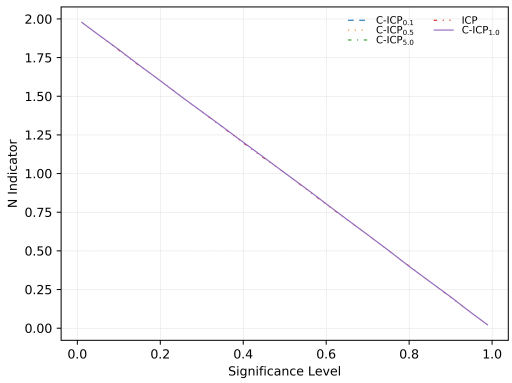


(e) Empty Predictions

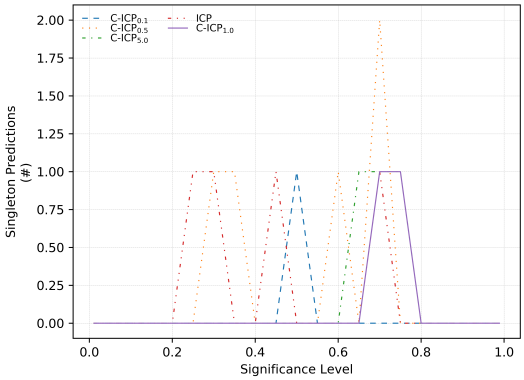
Figure A.13: Validity and efficiency measures obtained by C-ICP and ICP at different significance levels on the covertype data-set for the LR problem. C-ICP uses AVM as a heuristic for compressing the data for LR.



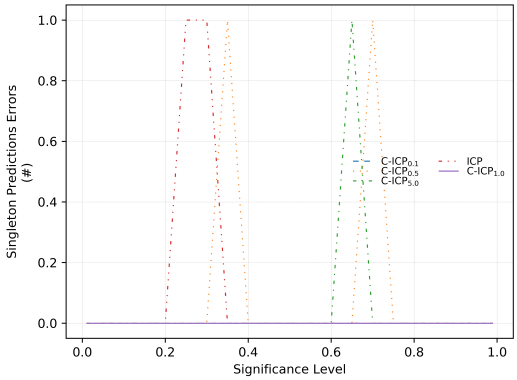
(a) Errors



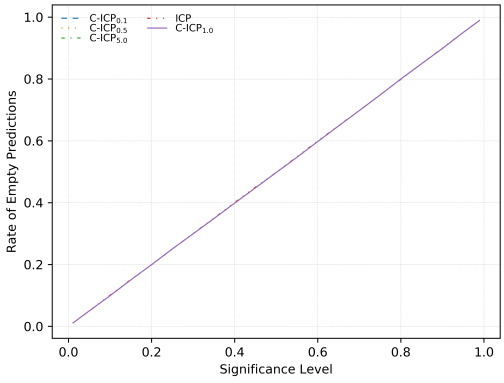
(b) Average Size of Prediction Sets (N Indicator)



(c) Singleton Predictions

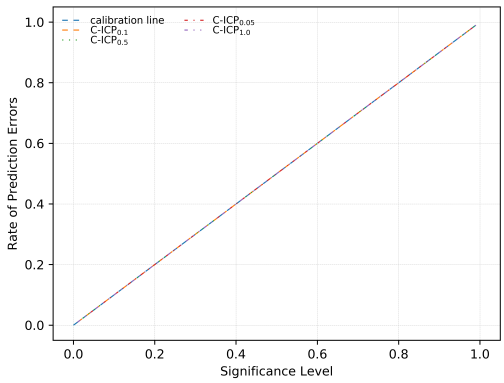


(d) Singleton Errors

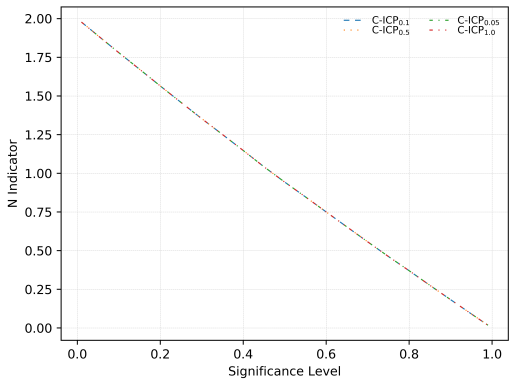


(e) Empty Predictions

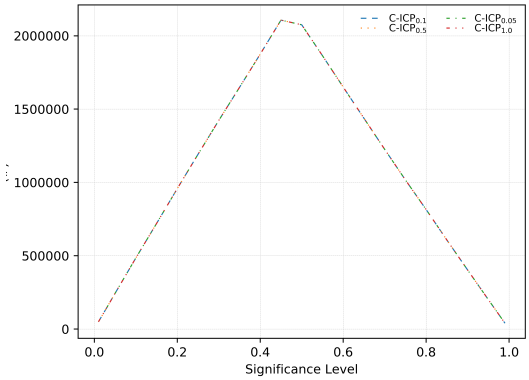
Figure A.14: Validity and efficiency measures obtained by C-ICP and ICP at different significance levels on the webspam data-set for the LR problem. C-ICP uses AVM as a heuristic for compressing the data for LR.



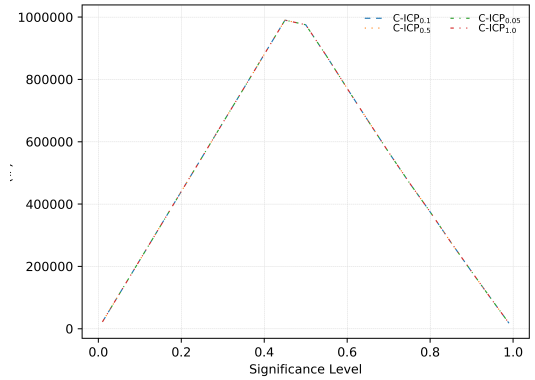
(a) Errors



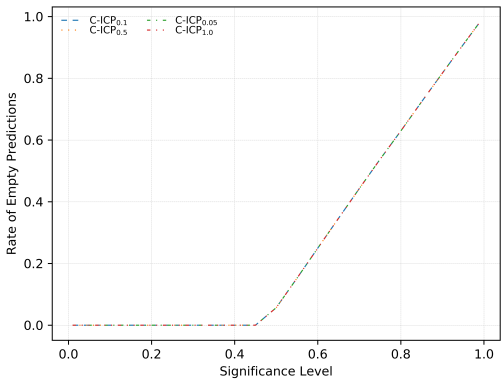
(b) Average Size of Prediction Sets (N Indicator)



(c) Singleton Predictions

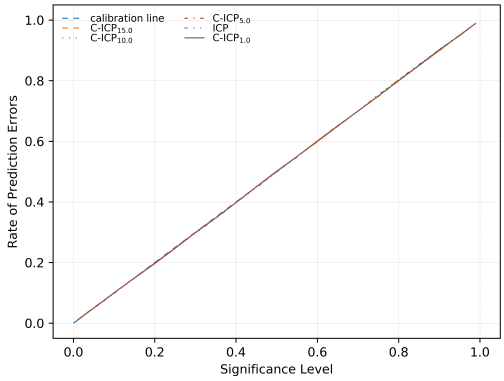


(d) Singleton Errors

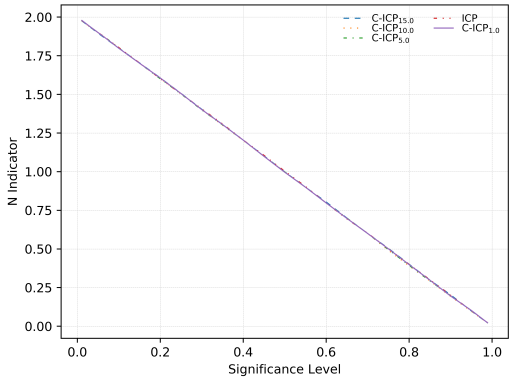


(e) Empty Predictions

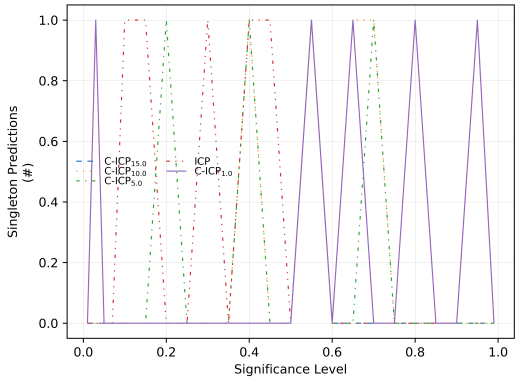
Figure A.15: Validity and efficiency measures obtained by C-ICP and ICP at different significance levels on the higgs data-set for the LR problem. C-ICP uses AVM as a heuristic for compressing the data for LR.



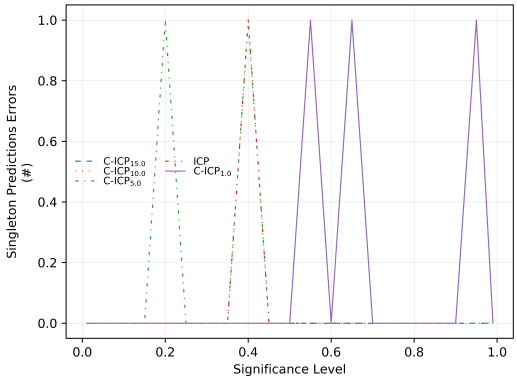
(a) Errors



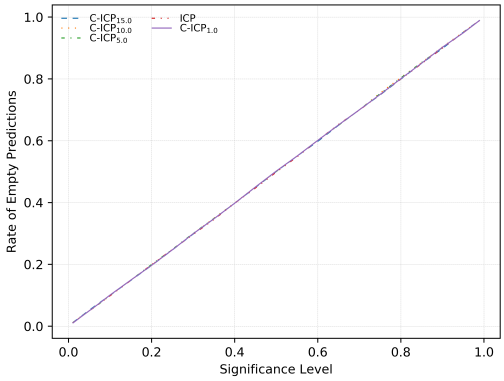
(b) Average Size of Prediction Sets (N Indicator)



(c) Singleton Predictions

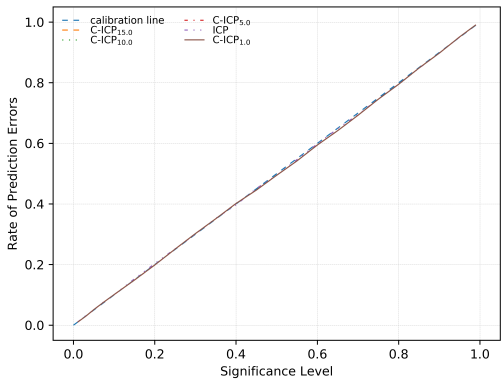


(d) Singleton Errors

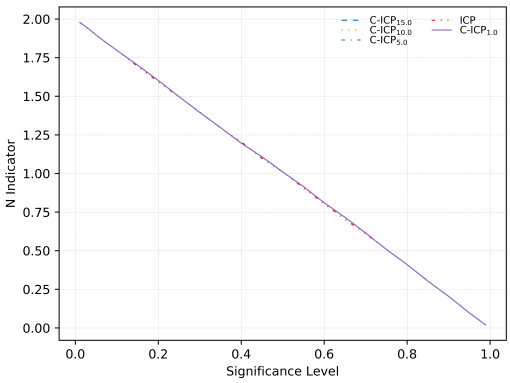


(e) Empty Predictions

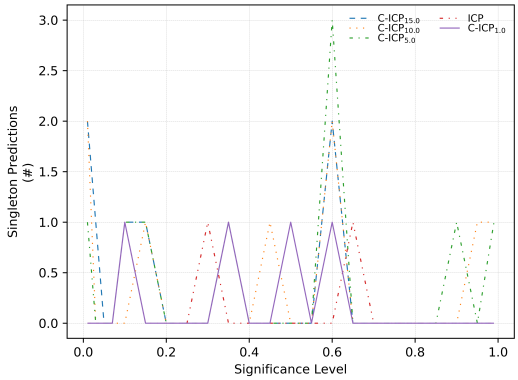
Figure A.16: Validity and efficiency measures obtained by C-ICP and ICP at different significance levels on the ijcnn1 data-set for the LR problem. C-ICP uses AVM as a heuristic for compressing the data for LR.



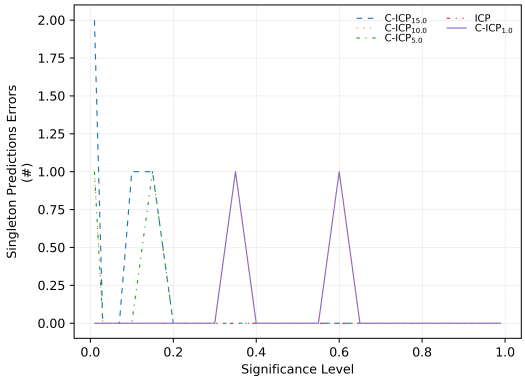
(a) Errors



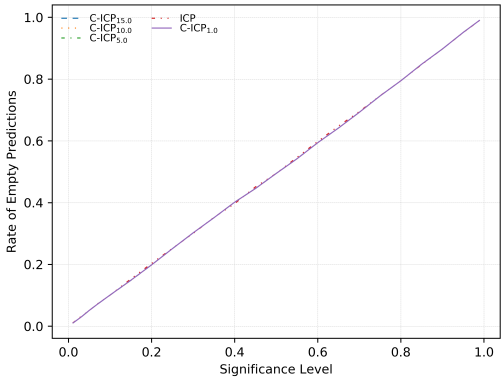
(b) Average Size of Prediction Sets (N Indicator)



(c) Singleton Predictions

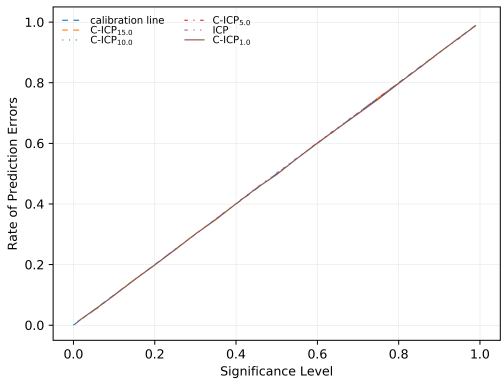


(d) Singleton Errors

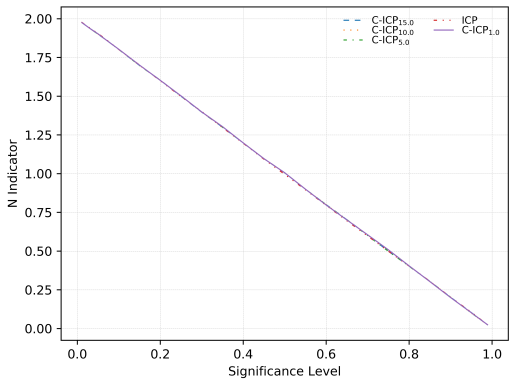


(e) Empty Predictions

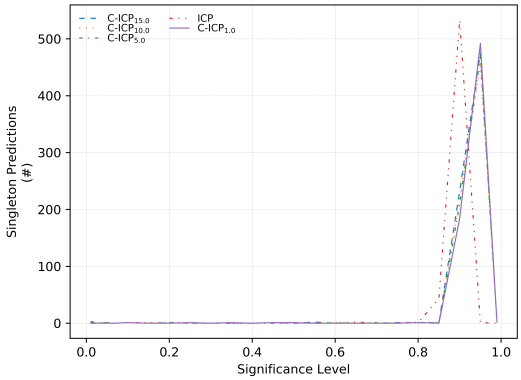
Figure A.17: Validity and efficiency measures obtained by C-ICP and ICP at different significance levels on the a9a data-set for the LR problem. C-ICP uses AVM as a heuristic for compressing the data for LR.



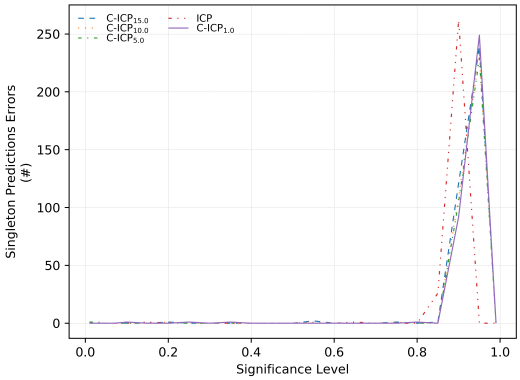
(a) Errors



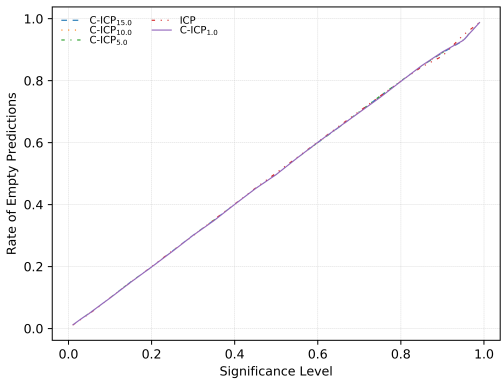
(b) Average Size of Prediction Sets (N Indicator)



(c) Singleton Predictions



(d) Singleton Errors



(e) Empty Predictions

Figure A.18: Validity and efficiency measures obtained by C-ICP and ICP at different significance levels on the w8a data-set for the LR problem. C-ICP uses AVM as a heuristic for compressing the data for LR.

Contents

Cor animalium, fundamentum eſt vitæ, princeps omnium, Microcoſmi Sol, a quo omnis vegetatio dependet, vigor omnis & robur emanat.

The heart of animals is the foundation of their life, the sovereign of everything within them, the sun of their microcosm, that upon which all growth depends, from which all power proceeds.

— William Harvey

B

Insight on the Sensitivity Measure using the RCA Algorithm

This appendix attempts to shed some light on the nature of sensitivities in the context of randomised algorithms for constructing coresets. We believe that the concept of sensitivity could be an interesting topic on its own right, and it could be of independent interest even beyond the confines of coresets. Since sensitivities, to the best of our knowledge, have not been empirically analysed nor explained in coreset works, mostly because these works are interested in theoretical aspects of coresets; we attempt here to show how the sensitivity captures the main structure of the input data. We also believe that this kind of analysis could serve as a starting point for linking sensitivities to labelling information in the input data. That is, we have seen in Chapters 4 and 6 that coresets are vulnerable to unbalancedness, and thus, we believe that an antidote to this problem could be to introduce labelling information in the compression process. In randomised approaches like the RCA algorithm, presented on Section 3.5.2 and used extensively on Chapters 4 and 6, this idea could mean to incorporate some of the unbalancedness information in the sensitivities scores. Thus

B.1 Covertype Sensitivity Structure

B.2 Webspam Sensitivity Structure

It is the mark of an educated mind to be able to entertain a thought without accepting it.

— Aristotle. *Metaphysics.*

C Complementary Experiments for IVAP-WEB

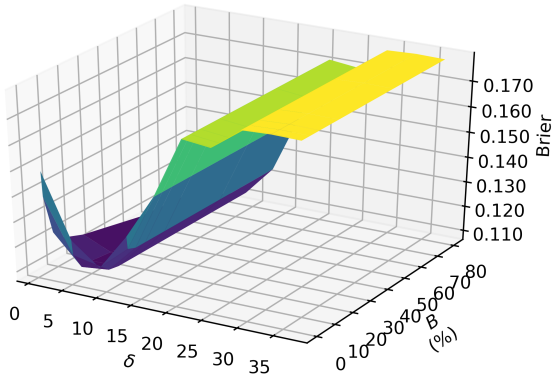
This appendix aims to shed some light on how the parameters of the AVM coreset-construction algorithm correlate to each other. For this demonstration, we have decided to use three datasets from our master table of data found in Chapter 2: ijcnn1, a9a and w8a. The rationale for this decision is strictly computational: their sizes are suitable for running many instances of avm with different parameter settings in order to demonstrate how they relate to each other. In principle, this relation should hold for every input data, although the values themselves will of course change from dataset to dataset.

We mentioned in Chapter 3 that AVM is non-parametric with respect to the number of core points. Theoretically, the algorithm is designed to deal with an *unbounded* number of input points, always creating a new ball whenever a new point does not fall inside the existing ones. In practice, however, one would want to have some control over the number of core points allowed on the system, specially because the model size of SVM, and hence the overall computing time, grows with each new core points. We use a parameter B, which is a positive integer specifying the maximum number of core points allowed in the coreset, to regulate AVM's trade-off between performance and computational efficiency. Then, if a new point

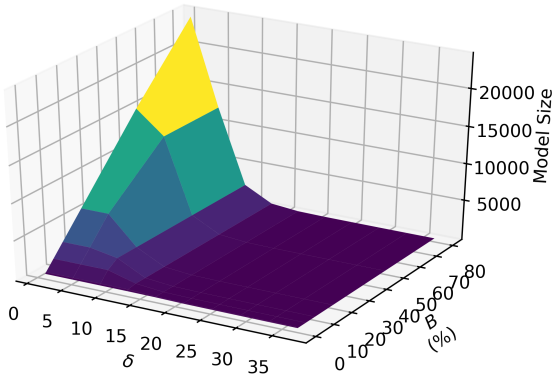
arrives and it does not fall inside the existing balls, we only construct a new ball if the number of core points, or equivalently the model size, is less or equal to B; if not, the point is simply discarded; we call such points *out-of-balls* (OOB) points. It is not hard to see that there must be an important relationship between the budget size B and the diameter of balls δ. For example, a small δ will cause the number of core points to increase rapidly, and hence the role of B will be more critical. On the other hand, large values of δ will make the model size grow very slowly, and it is likely that all the points will fall inside existing balls before B takes effect.

Figures C.1, C.2 and C.1 show in great detail how the interplay between these two parameters influence different aspects of the final outcome of IVAPs-WEB for the a9a, w8a and ijcnn1 datasets, respectively; in particular, we see how the Brier score, the model size, the number of OOB points and the computing time change with different values of δ and B. It is important to state that the values for B are represented as percentages of the proper training set; specifically, we consider budget sizes of 0.5%, 1%, 5%, 10%, 20%, 50% and 80% of the proper training set. For δ, we considered the following distances in *Euclidean space:* 1, 5, 9, 13, 17, 21, 25, 29, 33 and 37.

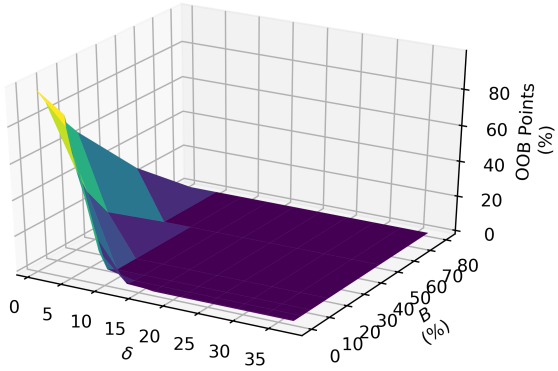
In the 3D plots, we can clearly see the fundamental trade-off between performance and computing efficiency which are typical of coreset-based methods: for both datasets, the peak performance can be achieved by setting δ very low and B very high; in this case, the balls are so small that each input point becomes a core point and hence the AVM degenerates into an instance of SGD for SVM. In other words, our coreset becomes as big as our original proper training set. However, if we look at the subplots showing the computing time (Sub-figures C.1d,C.2d and C.3d), we can clearly notice that the computing time becomes very high; this can be matched with the growth of the model size as well *i.e.* we can easily see that the time plots and the model-size plots are directly proportional. Furthermore, we can also see how using both small δ and B inevitably leads to a large number of out-of-ball points, making the overall computation really fast at the expense of worsening predictive performance (Brier score).



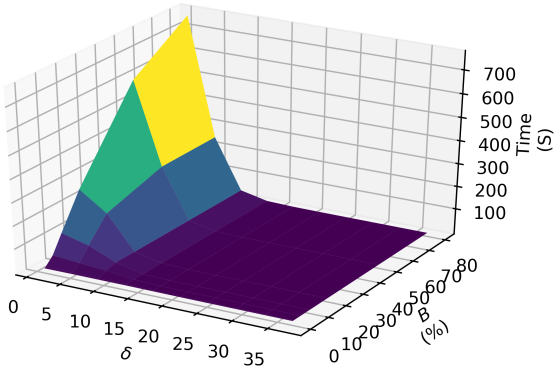
(a) Brier score for different diameter and budget-size values.



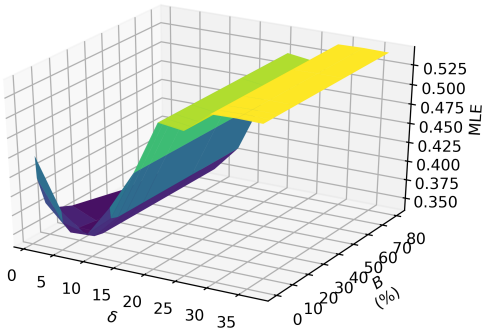
(b) Model size for different diameter and budget-size values.



(c) Disregarded (out-of-balls) points for different diameter and budget-size values.



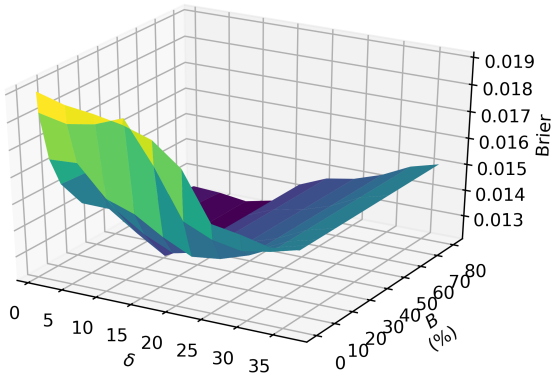
(d) Computing time for different diameter and budget-size values.



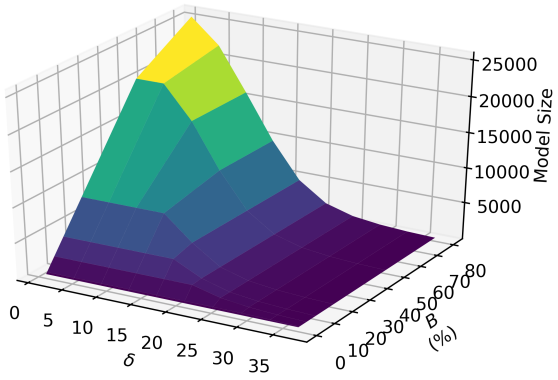
(e) The MLE score for different diameter and budget-size values.

Figure C.1: Interplay between δ and B for the a9a dataset.

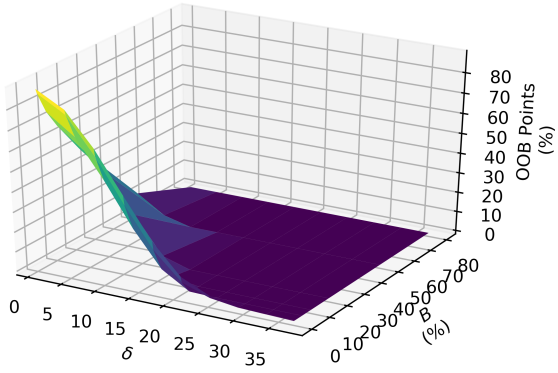
Figure C.2: Interplay between δ and B for the w8a dataset.



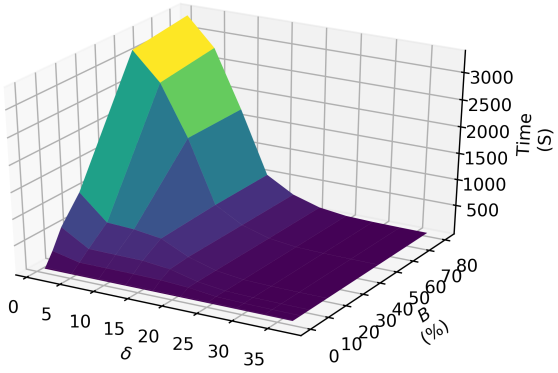
(a) Brier score for different diameter and budget-size values.



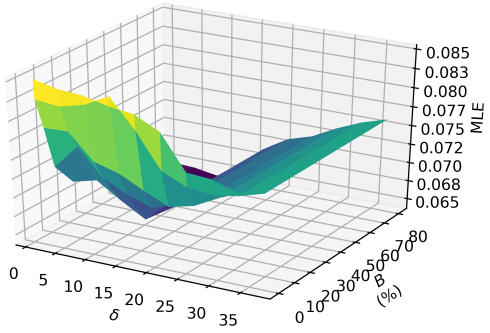
(b) Model size for different diameter and budget-size values.



(c) Disregarded (out-of-balls) points for different diameter and budget-size values.

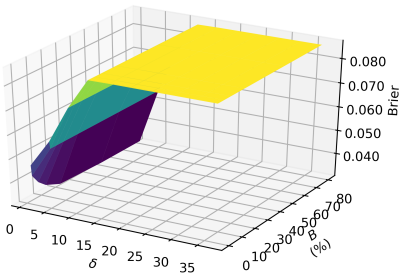


(d) Computing time for different diameter and budget-size values.

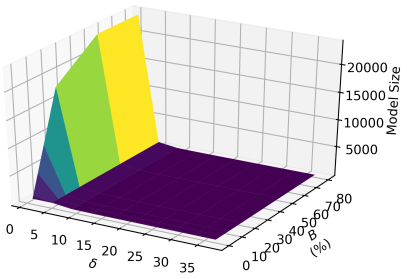


(e) The MLE score for different diameter and budget-size values.

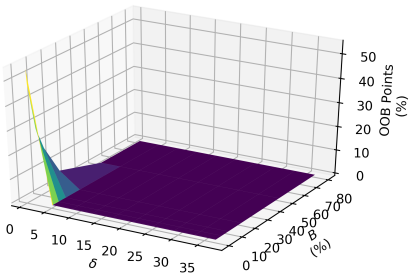
Figure C.3: Interplay between δ and B for the ijcnn1 dataset.



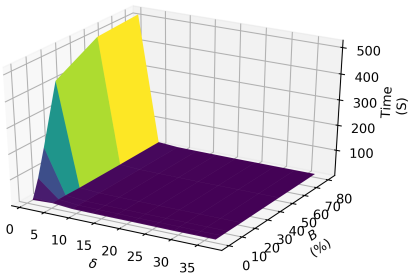
(a) Brier score for different diameter and budget-size values.



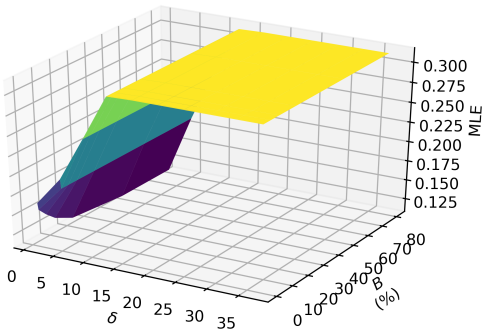
(b) Model size for different diameter and budget-size values.



(c) Disregarded (out-of-balls) points for different diameter and budget-size values.



(d) Computing time for different diameter and budget-size values.



(e) The MLE score for different diameter and budget-size values.

Wonder is the desire of knowledge.

— Thomas Aquinas.

D

The RDSF Technique with Different Regressors

The RDSF tehcnique allows for different regresssors to be used as the main mechanism for predicting the importance of the input points for the LR problem.In this appendix, we report the results obtained when using RDSF with different regressors in a plug-in/plug-out fashion. More concretely, we consider four different regression approaches: Ordinary Least Squares (OLS), Ridge Regression (RR), Lasso Regression (LSR) and Elastic Net (EN). To remind the reader, RR consists in fitting an OLS regressor with a L_2 regulariser; on the other hand, LSR trains an OLS regressor using a L_1 regulariser. Finally, EN finds an OLS regressor by using a convex combination of both L_1 and L_2 as its the regularisation term.

Depending of what regression algorithm was used to predict the sensitivities of input points, we can have one the following four RDSF instances:

- **RDFS-OLS:** an RDSF instance in which the input points' sensitivity scores were predicted using the ordinary-least-squares regression method. It is worth mentioning that this is the RDSF instance used in [118] and in Section 6.4.3.
- **RDFS-RR:** an instance of RDSF where the sensitivity for input points was predicted using the ridge regression method.

- **RDFS-LSR:** an RDSF instance that uses the lasso method for regression for predicting sensitivity scores.
- **RDFS-EN:** an instance of RDSF in which the sensitivity scores for input points are predicted via the elastic net regression method.

It is important to mention that for this study we only consider three metrics: Precision & Recall, AUROC and F1 Score. The reason for this decision already sheds the first lights on using different regressors for RDSF: different regressors give summaries of data that, when used to solve the LR problem, give classifiers that do not meaningfully differ in their accuracy score. However, we will see that when more involved performance metrics are considered, the difference in performance becomes more meaningful.

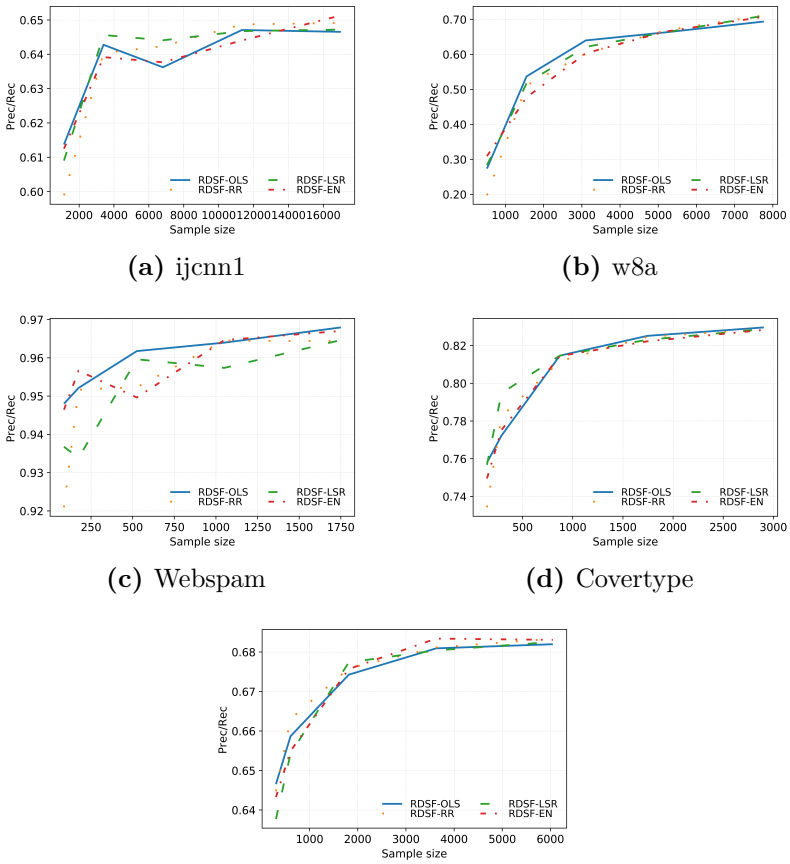
We also highlight that the point of this set of experiments is not to find out which of the RDSF instances considered is faster as the OLS method is by definition the fastest regression algorithm of all, and ridge regression is known to be faster than lasso. Instead, the aim of these experiments is to see whether more involved regression algorithms can help us obtain RDSF summaries that obtain better learning performance.

Figures D.1, D.2 and D.3 show the performance obtained when compressing the input data using different instances of the RDSF framework, specifically in terms of the precision & recall, AUROC and F1 score, respectively. Notice that these plots, as the ones found in the previous subsection, have the different sample sizes of the (RDSF) summaries on the x axis and the corresponding performance value on the y axis.

Quite surprisingly, RDSF trained with the simple OLS method seems to be the preferable approach for computing RDSF summaries in general: it is not always the best. but more often than not it gets really close to the best performing method. This is an enlightening result as OLS is the simplest method of the ones considered, and is seems to provide a very good explanation of the relationship between the input points and their sensitivities. Furthermore, we can make the conclusion

that regularisation does not help us much with the task of predicting sensitivities. If we also add up the fact that OLS has a closed-form solution and hence it is computationally very efficient, it becomes really hard to justify the use of any of the other regression algorithms to compress input data via the RDSF framework.

Finally, it is useful to mention that we also considered different sample sizes for the regression problem of sensitivity prediction *i.e.* different values for the parameter b in Algorithm 10; specifically, we tested setting b to 0.01%, 0.05%, 1%, 5%, 10% and 15% of the input data size; however, the behaviour observed is very similar to the results presented here.

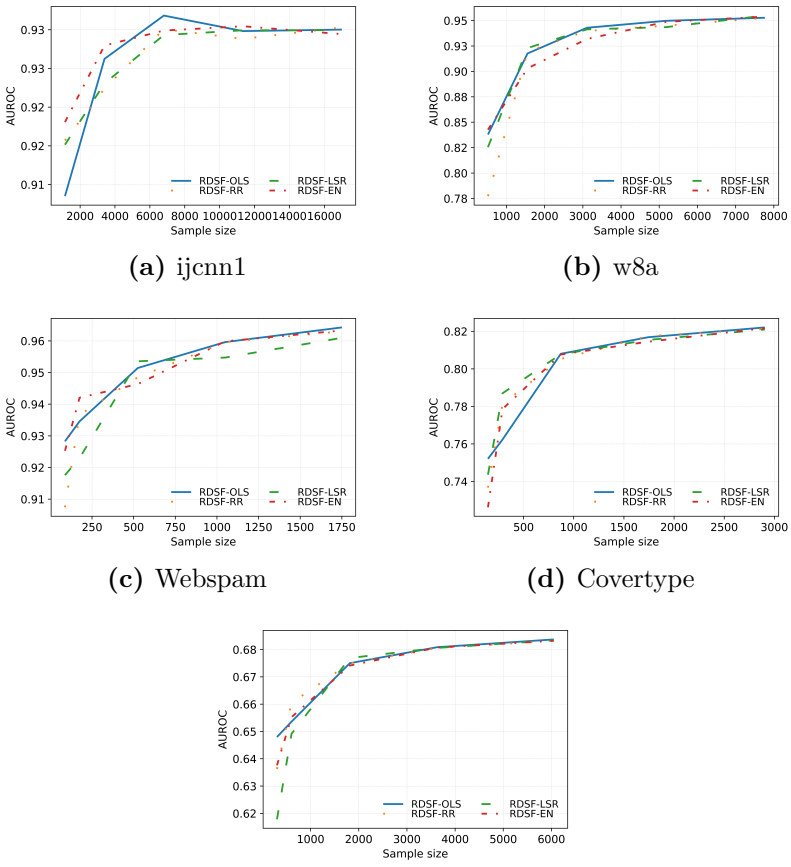


(a) ijcnn1 **(b)** w8a

(c) Webspam **(d)** Covertype

(e) Higgs

Figure D.1: Comparison of the areas under the precision & recall curve obtained from LR classifiers learned over RDSF summaries. The summaries were computed using different underlying regressors.

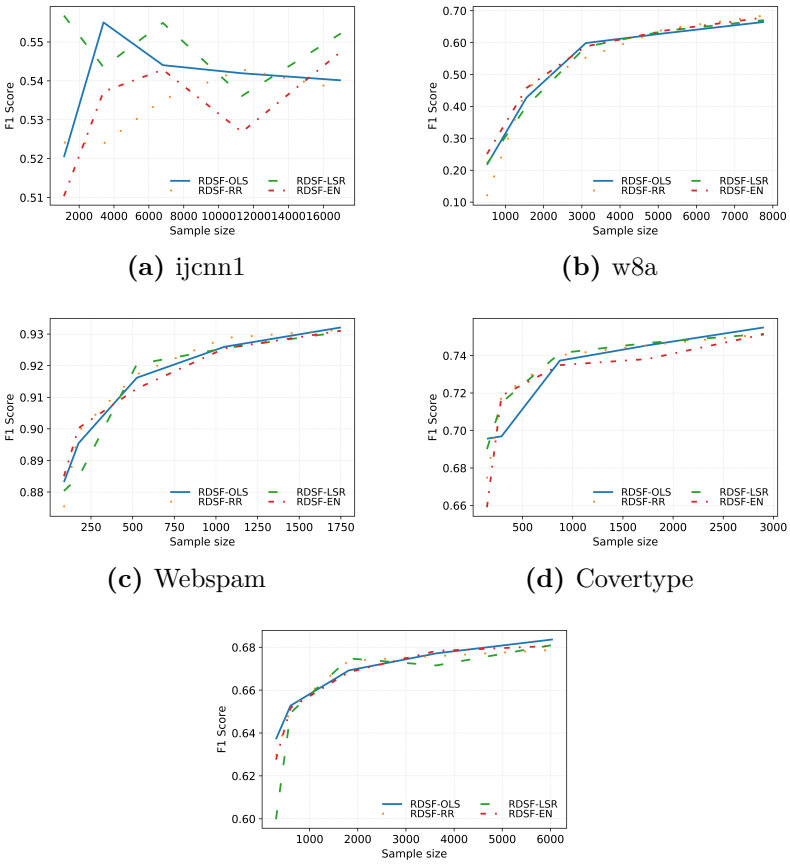


(a) ijcnn1 **(b)** w8a

(c) Webspam **(d)** Covertype

(e) Higgs

Figure D.2: Comparison of the AUROC obtained from LR classifiers learned over RDSF summaries. The summaries were computed using different underlying regressors.



(a) ijcnn1 **(b)** w8a

(c) Webspam **(d)** Covertype

(e) Higgs

Figure D.3: Comparison of the F1 scores obtained from LR classifiers learned over RDSF summaries. The summaries were computed using different underlying regressors.

References

[1] Mark Van Atten. "Two draft letters from Gödel on self-knowledge of reason". In: *Essays on Gödel's Reception of Leibniz, Husserl, and Brouwer*. Springer, 2015, pp. 157–162.

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

[3] Steve Webb, James Caverlee, and Calton Pu. "Introducing the Webb Spam Corpus: Using Email Spam to Identify Web Spam Automatically." In: *CEAS*. 2006.

[4] Asif Karim, Sami Azam, Bharanidharan Shanmugam, Krishnan Kannoorpatti, and Mamoun Alazab. "A comprehensive survey for intelligent spam email detection". In: *IEEE Access* 7 (2019), pp. 168261–168295.

[5] Ruben Buendia, Ola Engkvist, Lars Carlsson, Thierry Kogej, and Ernst Ahlberg. "Venn-Abers predictors for improved compound iterative screening in drug discovery". In: *Conformal and Probabilistic Prediction and Applications*. 2018, pp. 201–219.

[6] Dennis Bahler, Brian Stone, Carol Wellington, and Douglas W Bristol. "Symbolic, neural, and Bayesian machine learning models for predicting carcinogenicity of chemical compounds". In: *Journal of chemical information and computer sciences* 40.4 (2000), pp. 906–914.

[7] Shai Shalev-Shwartz, Yoram Singer, and Andrew Y Ng. "Online and batch learning of pseudo-metrics". In: *Proceedings of the twenty-first international conference on Machine learning*. 2004, p. 94.

[8] 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*. SIAM. 2010, pp. 1481–1489.

[9] Jerome H Friedman. *The elements of statistical learning: Data mining, inference, and prediction*. springer open, 2017.

[10] Tom Mitchell. *Machine learning*. McGraw hill Burr Ridge, 1997.

[11] Zoubin Ghahramani. "Unsupervised learning". In: *Summer School on Machine Learning*. Springer. 2003, pp. 72–112.

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

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

[14] Vladimir Vovk and Ivan Petej. "Venn-abers predictors". In: *arXiv preprint arXiv:1211.0025* (2012).

[15] Gianluca Zeni, Matteo Fontana, and Simone Vantini. "Conformal prediction: a unified review of theory and new challenges". In: *arXiv preprint arXiv:2005.07972* (2020).

[16] Vladimir Vovk, Ivan Petej, and Valentina Fedorova. "Large-scale probabilistic predictors with and without guarantees of validity". In: *Advances in Neural Information Processing Systems*. 2015, pp. 892–900.

[17] Ming Zhang, You Wang, Wei Zhang, Meng Yang, Zhiyuan Luo, and Guang Li. "Inductive conformal prediction for silent speech recognition". In: *Journal of neural engineering* 17.6 (2020), p. 066019.

[18] Martin Eklund, Ulf Norinder, Scott Boyer, and Lars Carlsson. "The application of conformal prediction to the drug discovery process". In: *Annals of Mathematics and Artificial Intelligence* 74.1 (2015), pp. 117–132.

[19] Ulf Norinder, Lars Carlsson, Scott Boyer, and Martin Eklund. "Introducing conformal prediction in predictive modeling. A transparent and flexible alternative to applicability domain determination". In: *Journal of chemical information and modeling* 54.6 (2014), pp. 1596–1603.

[20] Harris Papadopoulos, Volodya Vovk, and Alex Gammerman. "Conformal prediction with neural networks". In: *19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007)*. Vol. 2. IEEE. 2007, pp. 388–395.

[21] Antonis Lambrou, Harris Papadopoulos, and Alex Gammerman. "Evolutionary conformal prediction for breast cancer diagnosis". In: *2009 9th international conference on information technology and applications in biomedicine*. IEEE. 2009, pp. 1–4.

[22] Yitong Ren, Zhaojun Gu, Zhi Wang, Zhihong Tian, Chunbo Liu, Hui Lu, Xiaojiang Du, and Mohsen Guizani. "System log detection model based on conformal prediction". In: *Electronics* 9.2 (2020), p. 232.

[23] Pankaj K Agarwal, Sariel Har-Peled, and Kasturi R Varadarajan. "Geometric approximation via coresets". In: *Combinatorial and computational geometry* 52 (2005), pp. 1–30.

[24] Thomas Stearns Eliot. *The rock: A pageant play*. Houghton Mifflin Harcourt, 2014.

[25] Raymond E Wright. *Logistic regression.* American Psychological Association, 1995.

[26] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

[27] David Arthur and Sergei Vassilvitskii. "k-means++: The advantages of careful seeding". In: *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics. 2007, pp. 1027–1035.

[28] 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.

[29] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[30] Meelis Kull, Telmo M Silva Filho, and Peter Flach. "Beyond sigmoids: How to obtain well-calibrated probabilities from binary classifiers with beta calibration". In: *Electronic Journal of Statistics* 11.2 (2017), pp. 5052–5080.

[31] Nery Riquelme-Granada, Khuong Nguyen, and Zhiyuan Luo. "Coreset-based Conformal Prediction for Large-scale Learning". In: *Conformal and Probabilistic Prediction and Applications*. 2019, pp. 142–162.

[32] Fan Li and Yiming Yang. "A loss function analysis for classification methods in text categorization". In: *Proceedings of the 20th international conference on machine learning (ICML-03)*. 2003, pp. 472–479.

[33] Yaohua Tang, Weimin Guo, and Jinghuai Gao. "Efficient model selection for support vector machine with Gaussian kernel function". In: *2009 IEEE Symposium on Computational Intelligence and Data Mining*. IEEE. 2009, pp. 40–45.

[34] HW Corley. "Optimality conditions for maximizations of set-valued functions". In: *Journal of Optimization Theory and Applications* 58.1 (1988), pp. 1–10.

[35] Trung Le, Tu Dinh Nguyen, Vu Nguyen, and Dinh Phung. "Approximation vector machines for large-scale online learning". In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 3962–4016.

[36] Geoff Gordon and Ryan Tibshirani. "Karush-kuhn-tucker conditions". In: *Optimization* 10.725/36 (2012), p. 725.

[37] Dhruv Mahajan, S Sathiya Keerthi, S Sundararajan, and Léon Bottou. "A parallel SGD method with strong convergence". In: *arXiv preprint arXiv:1311.0636* (2013).

[38] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. "Pegasos: Primal estimated sub-gradient solver for svm". In: *Mathematical programming* 127.1 (2011), pp. 3–30.

[39] Olivier Chapelle. "Training a support vector machine in the primal". In: *Neural computation* 19.5 (2007), pp. 1155–1178.

[40] Zhuang Wang, Koby Crammer, and Slobodan Vucetic. "Breaking the curse of kernelization: Budgeted stochastic gradient descent for large-scale svm training". In: *Journal of Machine Learning Research* 13.Oct (2012), pp. 3103–3131.

[41] Ingo Steinwart. "Sparseness of support vector machines". In: *Journal of Machine Learning Research* 4.Nov (2003), pp. 1071–1105.

[42] *Dictionary by Merriam-Webster: America's most-trusted online dictionary*. URL: https://www.merriam-webster.com/.

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

[44] Ke Chen. "On coresets for k-median and k-means clustering in metric and euclidean spaces and their applications". In: *SIAM Journal on Computing* 39.3 (2009), pp. 923–947.

[45] Jieyang Chen, Nan Xiong, Xin Liang, Dingwen Tao, Sihuan Li, Kaiming Ouyang, Kai Zhao, Nathan DeBardeleben, Qiang Guan, and Zizhong Chen. "TSM2: optimizing tall-and-skinny matrix-matrix multiplication on GPUs". In: *Proceedings of the ACM International Conference on Supercomputing*. 2019, pp. 106–116.

[46] Marina Garcia de Lomana, Andrea Morger, Ulf Norinder, Roland Buesen, Robert Landsiedel, Andrea Volkamer, Johannes Kirchmair, and Miriam Mathea. "ChemBioSim: Enhancing Conformal Prediction of In Vivo Toxicity by Use of Predicted Bioactivities". In: *Journal of chemical information and modeling* 61.7 (2021), pp. 3255–3272.

[47] Ulf Norinder and Scott Boyer. "Conformal prediction classification of a large data set of environmental chemicals from ToxCast and Tox21 estrogen receptor assays". In: *Chemical research in toxicology* 29.6 (2016), pp. 1003–1010.

[48] Khuong An Nguyen. "A performance guaranteed indoor positioning system using conformal prediction and the WiFi signal strength". In: *Journal of Information and Telecommunication* 1.1 (2017), pp. 41–65.

[49] Mehdi Allahyari, Seyedamin Pouriyeh, Mehdi Assefi, Saied Safaei, Elizabeth D Trippe, Juan B Gutierrez, and Krys Kochut. "A brief survey of text mining: Classification, clustering and extraction techniques". In: *arXiv preprint arXiv:1707.02919* (2017).

[50] Dustin Mixon. "Short, fat matrices". In: *blog* (2013).

[51] H Brendan, Daniel Ramage, and Peter Richtárik. "Federated optimization: Distributed machine learning for on-device intelligence". In: *arXiv preprint arXiv:1610.02527* (2016).

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

[53] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. "Searching for exotic particles in high-energy physics with deep learning". In: *Nature communications* 5.1 (2014), pp. 1–9.

[54] John Platt. "Fast Training of Support Vector Machines Using Sequential Minimal Optimization". In: *Advances in Kernel Methods - Support Vector Learning*. MIT Press, Jan. 1998. URL: https://www.microsoft.com/en-us/research/publication/fast-training-of-support-vector-machines-using-sequential-minimal-optimization/.

[55] Danil Prokhorov. "IJCNN 2001 neural network competition". In: *Slide presentation in IJCNN* 1.97 (2001), p. 38.

[56] Sotiris Kotsiantis, Dimitris Kanellopoulos, Panayiotis Pintelas, et al. "Handling imbalanced datasets: A review". In: *GESTS International Transactions on Computer Science and Engineering* 30.1 (2006), pp. 25–36.

[57] John Von Neumann, V Tibor, et al. *The Neumann Compendium*. Vol. 1. World scientific, 1995.

[58] Refael Hassin and Asaf Levin. "A better-than-greedy approximation algorithm for the minimum set cover problem". In: *SIAM Journal on Computing* 35.1 (2005), pp. 189–200.

[59] Jan Karel Lenstra, David B Shmoys, and Éva Tardos. "Approximation algorithms for scheduling unrelated parallel machines". In: *Mathematical programming* 46.1 (1990), pp. 259–271.

[60] Oscar H Ibarra and Chul E Kim. "Fast approximation algorithms for the knapsack and sum of subset problems". In: *Journal of the ACM (JACM)* 22.4 (1975), pp. 463–468.

[61] Michael Sipser. "Introduction to the Theory of Computation". In: *ACM Sigact News* 27.1 (1996), pp. 27–29.

[62] Seyedali Mirjalili. "Genetic algorithm". In: *Evolutionary algorithms and neural networks*. Springer, 2019, pp. 43–55.

[63] Michel Gendreau and Jean-Yves Potvin. "Tabu search". In: *Search methodologies*. Springer, 2005, pp. 165–186.

[64] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

[65] Jeff M Phillips. "Coresets and sketches". In: *arXiv preprint arXiv:1601.00617* (2016).

[66] Pankaj K Agarwal, Cecilia M Procopiuc, and Kasturi R Varadarajan. "Approximation algorithms for k-line center". In: *European symposium on algorithms*. Springer. 2002, pp. 54–63.

[67] 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*. SIAM. 2013, pp. 1434–1453.

[68] Vladimir Braverman, Dan Feldman, and Harry Lang. "New Frameworks for Offline and Streaming Coreset Constructions". In: *CoRR* abs/1612.00889 (2016). arXiv: 1612.00889. URL: http://arxiv.org/abs/1612.00889.

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

[70] Sariel Har-Peled and Soham Mazumdar. "On coresets for k-means and k-medSian clustering". In: *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*. ACM. 2004, pp. 291–300.

[71] 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*. ACM. 2011, pp. 569–578.

[72] Dan Feldman and Leonard J Schulman. "Data reduction for weighted and outlier-resistant clustering". In: *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*. SIAM. 2012, pp. 1343–1354.

[73] Dan Feldman. "Core-sets: Updated survey". In: *Sampling Techniques for Supervised or Unsupervised Tasks* (2020), pp. 23–44.

[74] Gerhard Tutz and Harald Binder. "Boosting ridge regression". In: *Computational Statistics & Data Analysis* 51.12 (2007), pp. 6044–6059.

[75] Ozan Sener and Silvio Savarese. "Active learning for convolutional neural networks: A core-set approach". In: *arXiv preprint arXiv:1708.00489* (2017).

[76] Cenk Baykal, Lucas Liebenwein, Igor Gilitschenski, Dan Feldman, and Daniela Rus. "Data-dependent coresets for compressing neural networks with applications to generalization bounds". In: *arXiv preprint arXiv:1804.05345* (2018).

[77] Elad Tolochinsky and Dan Feldman. "Coresets for monotonic functions with applications to deep learning". In: *CoRR, abs/1802.07382* (2018).

[78] Jonathan Huggins, Trevor Campbell, and Tamara Broderick. "Coresets for scalable bayesian logistic regression". In: *Advances in Neural Information Processing Systems*. 2016, pp. 4080–4088.

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

[80] Sashank J Reddi, Barnabás Póczos, and Alexander J Smola. "Communication Efficient Coresets for Empirical Loss Minimization." In: *UAI*. 2015, pp. 752–761.

[81] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine learning* 20.3 (1995), pp. 273–297.

[82] Mihai Bădoiu and Kenneth L Clarkson. "Optimal core-sets for balls". In: *Computational Geometry* 40.1 (2008), pp. 14–22.

[83] Ivor W Tsang, James T Kwok, and Pak-Ming Cheung. "Core vector machines: Fast SVM training on very large data sets". In: *Journal of Machine Learning Research* 6.Apr (2005), pp. 363–392.

[84] Ivor W Tsang, Andras Kocsor, and James T Kwok. "Simpler core vector machines with enclosing balls". In: *Proceedings of the 24th international conference on Machine learning*. 2007, pp. 911–918.

[85] Yu-Chi Ho and David L Pepyne. "Simple explanation of the no-free-lunch theorem and its implications". In: *Journal of optimization theory and applications* 115.3 (2002), pp. 549–570.

[86] Olivier Bachem, Mario Lucic, and Andreas Krause. "Practical coreset constructions for machine learning". In: *arXiv preprint arXiv:1703.06476* (2017).

[87] Stephen Boyd, Lin Xiao, and Almir Mutapcic. "Subgradient methods". In: *lecture notes of EE392o, Stanford University, Autumn Quarter* 2004 (2003), pp. 2004–2005.

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

[89] Mihai Bādoiu, Sariel Har-Peled, and Piotr Indyk. "Approximate clustering via core-sets". In: *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. 2002, pp. 250–257.

[90] Sariel Har-Peled, Dan Roth, and Dav Zimak. "Maximum Margin Coresets for Active and Noise Tolerant Learning." In: *IJCAI*. 2007, pp. 836–841.

[91] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*. Vol. 87. Springer Science & Business Media, 2003.

[92] Michael B Cohen, Sam Elder, Cameron Musco, Christopher Musco, and Madalina Persu. "Dimensionality reduction for k-means clustering and low rank approximation". In: *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*. 2015, pp. 163–172.

[93] Sanjoy Dasgupta and Anupam Gupta. "An elementary proof of the Johnson-Lindenstrauss lemma". In: *International Computer Science Institute, Technical Report* 22.1 (1999), pp. 1–5.

[94] Deanna Needell, Rachel Ward, and Nati Srebro. "Stochastic gradient descent, weighted sampling, and the randomized Kaczmarz algorithm". In: *Advances in Neural Information Processing Systems*. 2014, pp. 1017–1025.

[95] Nabil H Mustafa and Kasturi R Varadarajan. "Epsilon-approximations and epsilon-nets". In: *arXiv preprint arXiv:1702.03676* (2017).

[96] Sashank J Reddi, Jakub Konečnỳ, Peter Richtárik, Barnabás Póczós, and Alex Smola. "AIDE: fast and communication efficient distributed optimization". In: *arXiv preprint arXiv:1608.06879* (2016).

[97] Sariel Har-Peled and Akash Kushal. "Smaller coresets for k-median and k-means clustering". In: *Discrete & Computational Geometry* 37.1 (2007), pp. 3–19.

[98] Marcel R Ackermann, Marcus Märtens, Christoph Raupach, Kamil Swierkot, Christiane Lammersen, and Christian Sohler. "StreamKM++: A cluste ing algorithm for data streams". In: *Journal of Experimental Algorithmics (JEA)* 17 (2012), pp. 2–4.

[99] Qi Zhang, Jinze Liu, and Wei Wang. "Approximate clustering on distributed data streams". In: *2008 IEEE 24th International Conference on Data Engineering*. IEEE. 2008, pp. 1131–1139.

[100] Dan Feldman, Andrew Sugaya, and Daniela Rus. "An effective coreset compression algorithm for large scale sensor networks". In: *2012 ACM/IEEE 11th International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE. 2012, pp. 257–268.

[101] Jon Louis Bentley and James B Saxe. "Decomposable searching problems I. Static-to-dynamic transformation". In: *Journal of Algorithms* 1.4 (1980), pp. 301–358.

[102] Charles Ziegler. "Edward Teller: Giant of the Golden Age of Physics. Stanley A. Blumberg, Louis G. PanosThe Advisors: Oppenheimer, Teller, and the Superbomb. Herbert F. YorkAtom and Void: Essays on Science and Community. J. Robert Oppenheimer". In: *Isis* 82.3 (1991).

[103] Glenn Shafer and Vladimir Vovk. "A tutorial on conformal prediction". In: *Journal of Machine Learning Research* 9.Mar (2008), pp. 371–421.

[104] Alexander Gammerman and Vladimir Vovk. "Hedging predictions in machine learning: The second computer journal lecture". In: *The Computer Journal* 50.2 (2007), pp. 151–163.

[105] Vladimir Vapnik. *Statistical learning theory. 1998*. Vol. 3. Wiley, New York, 1998.

[106] Vladimir Vovk. "Conditional validity of inductive conformal predictors". In: *Asian conference on machine learning*. 2012, pp. 475–490.

[107] JC Platt. *Probabilities for SV Machines, Advances in Large Margin Classifiers*. 1999.

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

[109] Valentina Fedorova, Alex Gammerman, Ilia Nouretdinov, and Vladimir Vovk. "Conformal prediction under hypergraphical models". In: *IFIP International Conference on Artificial Intelligence Applications and Innovations*. Springer. 2013, pp. 371–383.

[110] Ulf Johansson, Rikard König, Tuve Löfström, and Henrik Boström. "Evolved decision trees as conformal predictors". In: *Evolutionary Computation (CEC), 2013 IEEE Congress on*. IEEE. 2013, pp. 1794–1801.

[111] Nery Riquelme-Granada, Khuong An Nguyen, and Zhiyuan Luo. "Fast probabilistic prediction for kernel SVM via enclosing balls". In: *Conformal and Probabilistic Prediction and Applications*. PMLR. 2020, pp. 189–208.

[112] Gottfried Wilhelm Leibniz. *The monadology and other philosophical writings*. 1898.

[113] Miriam Ayer, H Daniel Brunk, George M Ewing, William T Reid, and Edward Silverman. "An empirical distribution function for sampling with incomplete information". In: *The annals of mathematical statistics* (1955), pp. 641–647.

[114] Chih-Chung Chang and Chih-Jen Lin. "LIBSVM: A library for support vector machines". In: *ACM Transactions on Intelligent Systems and Technology* 2 (3 2011). Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm, 27:1–27:27.

[115] John Platt. "Sequential minimal optimization: A fast algorithm for training support vector machines". In: *Microsoft Research Technical Report* (1998).

[116] Alan M Turing. "Computing machinery and intelligence". In: *Parsing the turing test*. Springer, 2009, pp. 23–65.

[117] T Velmurugan and T Santhanam. "Computational complexity between K-means and K-medoids clustering algorithms for normal and uniform distributions of data points". In: *Journal of computer science* 6.3 (2010), p. 363.

[118] N. Riquelme-Granada, Khuong An Nguyen., and Zhiyuan Luo. "On Generating Efficient Data Summaries for Logistic Regression: A Coreset-based Approach". In: *Proceedings of the 9th International Conference on Data Science, Technology and Applications - Volume 1: DATA,* INSTICC. SciTePress, 2020, pp. 78–89.

[119] Gary C McDonald. "Ridge regression". In: *Wiley Interdisciplinary Reviews: Computational Statistics* 1.1 (2009), pp. 93–100.

[120] J Ranstam and JA Cook. "LASSO regression". In: *Journal of British Surgery* 105.10 (2018), pp. 1348–1348.

[121] Hui Zou and Trevor Hastie. "Regression shrinkage and selection via the elastic net, with applications to microarrays". In: *JR Stat Soc Ser B* 67 (2003), pp. 301–20.

[122] Jesse Davis and Mark Goadrich. "The relationship between Precision-Recall and ROC curves". In: *Proceedings of the 23rd international conference on Machine learning*. 2006, pp. 233–240.

[123] Cyril Goutte and Eric Gaussier. "A probabilistic interpretation of precision, recall and F-score, with implication for evaluation". In: *European Conference on Information Retrieval*. Springer. 2005, pp. 345–359.

[124] Shai Shalev-Shwartz et al. "Online learning and online convex optimization". In: *Foundations and Trends® in Machine Learning* 4.2 (2012), pp. 107–194.

[125] Daniela Brauckhoff, Kave Salamatian, and Martin May. "Applying PCA for traffic anomaly detection: Problems and solutions". In: *IEEE INFOCOM 2009*. IEEE. 2009, pp. 2866–2870.

[126] FESL Husson, Sébastien Lê, and Jérome Pagès. "Variability of the representation of the variables resulting from PCA in the case of a conventional sensory profile". In: *Food quality and preference* 18.7 (2007), pp. 933–937.

[127] Laurens Van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE." In: *Journal of machine learning research* 9.11 (2008).

[128] Paulo E Rauber, Alexandre X Falcão, Alexandru C Telea, et al. "Visualizing Time-Dependent Data Using Dynamic t-SNE." In: (2016).

[129] Andrej Gisbrecht, Alexander Schulz, and Barbara Hammer. "Parametric nonlinear dimensionality reduction using kernel t-SNE". In: *Neurocomputing* 147 (2015), pp. 71–82.

[130] Dmitry Kobak and Philipp Berens. "The art of using t-SNE for single-cell transcriptomics". In: *Nature communications* 10.1 (2019), pp. 1–14.

[131] Angelos Chatzimparmpas, Rafael M Martins, and Andreas Kerren. "t-visne: Interactive assessment and interpretation of t-sne projections". In: *IEEE transactions on visualization and computer graphics* 26.8 (2020), pp. 2696–2714.

[132] Oded Goldreich. *Introduction to property testing*. Cambridge University Press, 2017.

[133] Oded Goldreich. "Property Testing". In: *Lecture Notes in Comput. Sci* 6390 (2010).

[134] Rich Caruana and Alexandru Niculescu-Mizil. "An empirical comparison of supervised learning algorithms". In: *Proceedings of the 23rd international conference on Machine learning*. 2006, pp. 161–168.

[135] Vladimir Vovk. "Cross-conformal predictors". In: *Annals of Mathematics and Artificial Intelligence* 74.1 (2015), pp. 9–28.