Proceedings of Machine Learning Research 128:1–20, 2020 Conformal and Probabilistic Prediction and Applications

Fast Probabilistic Prediction for Kernel SVM via Enclosing Balls

Nery Riquelme-Granada NERY.RIQUELMEGRANADA.2016@LIVE.RHUL.AC.UK
Khuong An Nguyen KHUONG.NGUYEN@RHUL.AC.UK
Zhiyuan Luo ZHIYUAN.LUO@RHUL.AC.UK
Department of Computer Science, Royal Holloway University of London
Egham, Surrey TW20 0EX, United Kingdom

Editor: Alexander Gammerman, Vladimir Vovk, Zhiyuan Luo, Evgueni Smirnov and Giovanni Cherubin

Abstract

Support Vector Machine (SVM) is a powerful paradigm that has proven to be extremely useful for the task of classifying high-dimensional objects. It does not only perform well in learning linear classifiers, but also shows outstanding performance in capturing non-linearity through the use of *kernels*. In principle, SVM allows us to train "scoring" classifiers *i.e.* classifiers that output a prediction score. However, it can also be adapted to produce probability-type outputs through the use of the Venn-Abers framework. This allows us to obtain valuable information on the labels distribution for each test object. This procedure, however, is restricted to very small data given its inherent computational complexity. We circumvent this limitation by borrowing results from the field of computational geometry. Specifically, we make use of the concept of a coreset: a small summary of data that is constructed by discretising the input space into *enclosing balls*, so that each ball will be represented by only one object. Our results indicate that training Venn-Abers predictors using enclosing balls provides an average acceleration of 8 times compared to the regular Venn-Abers approach while largely retaining probability calibration. These promising results imply that we can still enjoy well-calibrated probabilistic outputs for kernel SVM even in the realm of large-scale datasets.

Keywords: Kernel SVM, Venn-Abers prediction, Enclosing Balls, Coresets.

1. Introduction

The term "big data" has become increasingly relevant in the last decade. As indicated in Feldman et al. (2013), people created 2.5×10^{18} bytes of data every day in 2012. This poses an interesting situation for the field of machine learning: on one hand, more data means potentially higher chances of discovering meaningful properties about the underlying data-generating mechanism; on the other hand, huge input sizes usually imply much more computational effort *e.g.* computing time, storage, etc, spent on learning from the data. Hence, learning algorithms should be able, in one way or another, to cope with this challenging scalability problem.

One particular learning paradigm that enjoys great success in pattern classification is that of *Support Vector Machine* (SVM) (Vapnik (2013)), which finds *large-margin* separators (*a.k.a.* half-spaces) that not only keep training points on the correct side of the half-space, but also keep them *far* from it (Shalev-Shwartz and Ben-David (2014)). The predictive capacity of SVM can be greatly enhanced by the use of *kernels*: mappings to high-dimensional feature spaces that allow SVM 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.

Using the above paradigm, we obtain a binary classifier which can be used on a test object x_{n+1} to obtain a real-valued *score* that will hopefully allow us to pick the correct label for x_{n+1} from a set of possible labels *e.g.* {-1,1}. This kind of result characterises what is usually referred as a *scoring classifier*.

Despite their popularity, scoring classifiers do not usually tell us much about the probability distributions of the labels a test object can have as their nature is non-probabilistic. We can, however, obtain well calibrated probabilities from these scoring classifiers through *Venn-Abers Prediction* (Vovk and Petej (2012)): a learning framework where virtually any scoring classifier can be used as a sub-routine to produce probability-type results. We are particularly interested in *Inductive Venn-Abers Prediction*[1] (IVAP), a variant of Venn-Abers Prediction where we rely on the use of a calibration set. We can use IVAP to translate all the predictive power of the SVM paradigm into valuable probabilistic information. This great advantage, however, does not come for free: training SVM with kernel models under the IVAP framework is computationally expensive and hence its uses are usually restricted to relatively small data.

To mitigate the above computational problem, we propose to indirectly speed up the Venn-Abers procedure by accelerating the underlying SVM procedure; specifically, we will train the SVM classifier on a small summary of the full input data, which will allow the learning algorithm to converge considerably faster. Hence, our approach accelerates the computation of a SVM classifier, and consequently the computation of IVAPs, without changing any of these methods; that is, we simply work with the input data, leaving the learning algorithms untouched. Our procedure can be seen as an implementation of the relatively new paradigm of *sketch and solve* (Munteanu and Schwiegelshohn (2018)). In our case, the small summary of input data is a *coreset* (Phillips (2016)): a small set that provably correctly approximates a big set. The idea is to construct a coreset for our input data, the *sketch* part; and then train a kernel SVM classifier with IVAP, the *solve* part. We shall empirically show that the proposed approach saves substantial computing time while maintaining the predictive performance of SVM. Thus, all the computing time we save comes from speeding up the SVM training time.

We outline our contributions as follows:

- a very fast sketch-and-solve solution to probabilistic kernel SVM prediction via the Venn-Abers framework;
- empirical evaluations to show that our method allows Venn-Abers prediction to scale up to large datasets for the problem of kernel SVM;
- shedding light on the interaction between coresets and techniques from reliable machine learning.

1. Note that throughout the paper IVAP refers to *Inductive Venn-Abers Prediction i.e.* the learning framework, while IVAPs refers to *Inductive Venn-Abers Predictors i.e.* the predictors themselves.

The rest of the paper is structured as follows: Section 2 introduces our notation, the SVM paradigm and the Venn-Abers framework. Section 3 defines coresets and introduces our proposed method. Section 4 discusses the experiments and presents our empirical results; and Section 5 concludes the paper.

2. Background

This section overviews the theoretical background of SVM and IVAP, which we will incorporate in our enclosing balls approach later on.

2.1. The SVM Approach

We are interested in performing binary data classification. We assume our input data is of the form $\{(x_i, y_i)\}_{i=1}^n$, where $x_i \in \mathrm{X}$ and $y_i \in \{-1, 1\}$ are an *object* and its *label*, respectively. Let $\mathrm{X} \subseteq \mathbb{R}^d$. The SVM paradigm allows us to cast object classification under diverse assumptions and different formulations; in our work we consider the *soft-SVM* formulation in its *primal* form [2], which formally can be defined as minimising the following function (Shalev-Shwartz and Ben-David (2014)):

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

where $\ell(w; x_i, y_i) := \mathbf{max}\{0, 1 - y_i(\langle w, x_i \rangle)\}$ is called the *hinge loss* and λ is the parameter determining the trade-off between increasing the margin size and ensuring that training data lie on the correct side of the margin. Hence, the optimal SVM solution is defined as:

$$w^* := \arg\min_{w} f(w) \tag{2}$$

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

As mentioned before, we can greatly enhance the generalisation capabilities of the SVM solution in (2) by incorporating the use of *kernels*. Let $K(x, x') := \exp(-\gamma ||x - x'||^2)$ and $\phi(\cdot)$ be the *radial basis function* (RBF) kernel and the feature mapping induced by the kernel, respectively. Notice that we can represent the kernel through its induced feature mapping *i.e.* $K(x, x') := \langle \phi(x), \phi(x') \rangle$. We can then re-write (1) as

$$f(w) := \lambda ||w||^2 + \frac{1}{n} \sum_{i=1}^{n} \mathbf{max}\{0, 1 - y_i(\langle w, \phi(x_i) \rangle)\} \tag{3}$$

which now applies the mapping ϕ to all x_i, giving us the *kernel-SVM* (KSVM) objective function.

2. We need the primal SVM formulation as we are interested in using a gradient-based solver as part of our proposed method.

2.2. Scoring Classifiers and Probabilities

The SVM formulation we just defined gives a hyperplane (defined by w^*) [3] and in the most basic classification scenario, we want to use it to classify a test object, x_{n+1}. To do so, we need to compute the scalar $\langle w^*, x_{n+1}\rangle$ and then typically use some threshold to do the actual label prediction *i.e.* $sign(\langle w^*, x_{n+1}\rangle)$ where $sign(a)$ outputs 1 if $a > 0$ and -1 otherwise. This is an example of *scoring classification* (Vovk and Petej (2012)) where the *score* is the scalar $\langle w^*, x_{n+1}\rangle$. Hence, we call SVM, and any other method that produces this kind of result, a *scoring classifier*.

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 2.4, this extra information comes at a high computational cost.

2.3. Inductive Venn-Abers Prediction

The Venn-Abers method[4] is a high-level learning framework designed to transform the output of scoring classifiers into probabilistic ones. It relies on the method of *Isotonic Regression* to calibrate the prediction scores obtained by a scoring classifier. For a more compact notation, let $s : \mathrm{X} \to \mathbb{R}$ be the *scoring function* that characterises a scoring classifier *i.e.* in our case, this function could be of the form $s(x) := \langle w^*, x\rangle$, where w^* 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 (Vovk et al. (2015)):

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

(ii) Train a scoring classifier on the proper training set.

(iii) Find the scores $s_1, s_2, \ldots, s_k$ of the calibration objects $x_1, x_2, \ldots, x_k$.

(iv) For a test object x, compute its score s.

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

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

3. For the sake of exposition, assume the bias term b is included in w^*.

4. "Venn" because the method is a form of *Venn Machine* (Vovk et al. (2005)), "Abers" after the surnames' initials of the authors in Ayer et al. (1955), which proposed the underlying technique used in the method.

predicting the positive label. Hence, ideally, we would like p_0 and p_1 to be close to each other.

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 4.1 that there are well-defined procedures for turning the pair (p_0, p_1) into a point-probability p.

2.4. The Computational Cost

We have defined our learning problem of interest in Section 2.1 and we then presented IVAP, an approach to transform the prediction scores obtained by KSVM (or any scoring classifier) into probability-type results. We now look into the central issue of our work: computational efficiency. Even though IVAPs are the most computationally efficient members of the Venn-Abers family, scaling them, and the closely related conformal predictors, to modern large-scale datasets is an active area of research. Specifically, much effort have been put in accelerating Venn-Abers (see Vovk et al. (2015), Vovk and Petej (2012) and Buendia et al. (2018)). Our approach follows a different philosophy: we propose to use a sketch-and-solve approach which means that we want to accelerate IVAPs by carefully reducing the input data, leaving the algorithm *untouched*. A similar approach has been applied to conformal prediction in Riquelme-Granada et al. (2019).

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, we could accelerate the overall IVAP process by accelerating the training of our KSVM algorithm.

Consider training our KSVM classifier, for example, using the widely popular LIBSVM algorithm (Chang and Lin (2011)), which is the most famous implementation of Platt's *Sequential Minimal Optimization* (SMO) algorithm (Platt (1998b)). Even though we are guaranteed to obtain extremely high quality solutions for kernel 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 datasets.

A reasonable alternative to LIBSVM is to attempt to train our underlying scoring classifier using the approach of *Stochastic Gradient Descent* (SGD) Mahajan et al. (2013), 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 (Konecnỳ et al. (2016)).

In the next Section, we discuss yet another alternative; a relatively new method to solve KSVM in sketch-and-solve fashion: *Approximation Vector Machines* (AVM) (Le et al. (2017)); an approach that is strongly based on SGD, but with the crucial difference that it learns over coresets: small summaries of the input data that allow us to provably correctly approximate the optimal KSVM classifier w^*, while preventing computational overheads.

3. IVAP-WEB: Inductive Venn-Abers Prediction With Enclosing Balls

As anticipated, we address the computational aspect of the Venn-Abers framework by embracing the relatively new computational paradigm of *sketch and solve* (Munteanu and Schwiegelshohn (2018)), which consists in improving the computing time of algorithms, not by re-designing the algorithms or even replacing them for faster ones, but by reducing the input sizes. Hence, the same potentially inefficient algorithm will surely terminate faster over reduced data.

The above approach can be naturally cast in our machine learning context. That is, we could attempt to reduce the input data for our learning algorithm for the sake of computational efficiency. However, restricting our learners from data seems counterproductive, and even dangerous, as we know that in machine learning more data implies better predictive generalisation. Hence, we require more than simply reducing the input size; we need to have some sort of *theoretical guarantee* that will protect our learning algorithm from arbitrary decreases in predictive power.

Coresets (Braverman et al. (2016), Riquelme-Granada et al. (2020)) provide a well-established framework for reducing the input size for a learning algorithm and, at the same time, largely retaining its generalisation capability.

Formally, let function f be the objective function of some learning problem and let $\mathcal{D}$ be the input data. Then, we say that $\mathcal{C}$ is an ϵ-coreset for $\mathcal{D}$ if the following condition holds:

$$|f(\mathcal{D}) - f(\mathcal{C})| \leq \epsilon f(\mathcal{D}) \tag{4}$$

where ϵ is the error parameter. This expression establishes the main error bound offered by coresets. Hence, coresets are lossy compressed versions of the input data $\mathcal{D}$ and the amount of information loss is quantified by ϵ. Note that we generally need $|\mathcal{C}| \ll |\mathcal{D}|$. Thus, coresets give us a systematic framework under which we can reduce or *compress* our input data by keeping the most important data points with respect to f. The question now is how to construct such set $\mathcal{C}$, and the answer depends on the exact definition of the objective function f. In our case, we already know what f looks like *i.e.* Equation (3). The AVM algorithm, discussed at length in the next section, will give us a reliable framework for computing $\mathcal{C}$ from the input data, which for IVAPs, are the points in the proper training set. Furthermore, we will see that learning KSVM via SGD over such data compression will protect us from obtaining arbitrarily bad classifiers; this will be materialised in the form of a theoretical guarantee of the kind stated in (4).

3.1. Approximation Vector Machines

Approximation vector machines (Le et al. (2017)) can powerfully aid us in our task of efficiently training good-quality KSVM classifiers on very tiny portions of the input data, with the final goal of performing a sketch-and-solve kind of learning using IVAP. Once more, it is extremely important to clarify that without exceptions, the input points considered in our discussions are the points in the proper training set.

As mentioned before, AVM can be described as a coreset-based implementation of the well-known SGD solver. In other words, what Le *et al.* proposed in Le et al. (2017) is to exploit the sequential nature of SGD to incrementally build a KSVM model $\bar{w}_t$, $\forall t \in (1, \ldots, T)$, over *representative input points*. Furthermore, they proved that the final AVM solution, $\bar{w}_T$ is not far from the optimal KSVM solution, w^*.

Before describing AVM, it will be very helpful to review the below standard SGD steps. At iteration t, SGD performs the following operations:

(i) get (x_i, y_i).

(ii) set the learning rate η_t *e.g.* $\eta_t = 1/\lambda t$ (as recommended in Shalev-Shwartz et al. (2011)), where λ is the regularisation parameter in (1).

(iii) set the update for t as $g_t = \lambda w_t + \nabla\ell(w_t; x_i, y_i)$, where $\nabla\ell(w_t; x_i, y_i)$ is the gradient of the function ℓ, evaluated at (x_i, y_i).

(iv) perform the update $w_{t+1} = w_t - \eta_t g_t$.

Concentrating on step (iii) above, notice that for the hinge loss it is true that the gradient can be represented as $\nabla\ell(w_t; x_i, y_i) := \alpha_t \phi(x_i)$, where α_t is a scalar and ϕ is our mapping to the feature space. 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, $w_t := \sum_{i=1}^{t} \alpha_i^{(t)} \phi(x_i)$.

SGD would need to perform the above four steps for each of our input points in order to have a high-quality KSVM solution. However, when using kernels, the number of support vectors grows almost linearly with the number of input points; this phenomenon is commonly known as the *curse of kernelisation* (Wang et al. (2012)), and it poses a significant computational challenge that cannot be overlooked.

AVM allows us to mitigate the curse-of-kernelisation issue by only increasing the model size when the newly seen point is *far* from some stored representative 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 input space, a δ-coverage for X is defined as below.

Definition 1 (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 2. 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 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.

Equipped with all of the above discussion, we can appreciate the full AVM method in Algorithm 1. It is easy to see the SGD structure in the algorithm; 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 $\bar{w}_T$ such that $\mathbb{E}[|f(\bar{w}_T) - f(w^*)|] \leq \epsilon$, which in expectation gives us the coreset guarantee as in (4). For the theoretical proof on AVM's convergence we refer the reader to Le et al. (2017), pages 10, 11 and 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, in our experiments, we impose a *budget size* for AVM in order to control the size of the coreset.

```
**Input:** $\mathcal{D}$: input data, $\lambda$: regularisation parameter, $\mathcal{P} := (P_i)_{i \in I}$: $\delta$-coverage
**Output:** $w$: large-margin separator
initialise
  $w_1 \leftarrow 0$
  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 $x_t \in P_{it}$
      $g_t = \lambda w_t + \alpha_t \phi(c_{it})$
      $w_{t+1} = w_t - \eta_t g_t$
end
return $w_{T+1}$
```

Algorithm 1: The AVM algorithm proposed by Le *et al.* Notice that SGD is now expressed in terms of α_t and $\phi()$.

```
$\mathcal{P} \leftarrow \emptyset$
$M \leftarrow 0$
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: An algorithm for constructing $\mathcal{P}$, as defined in Le et al. (2017). Each P_i is a ball with radius $\delta/2$.

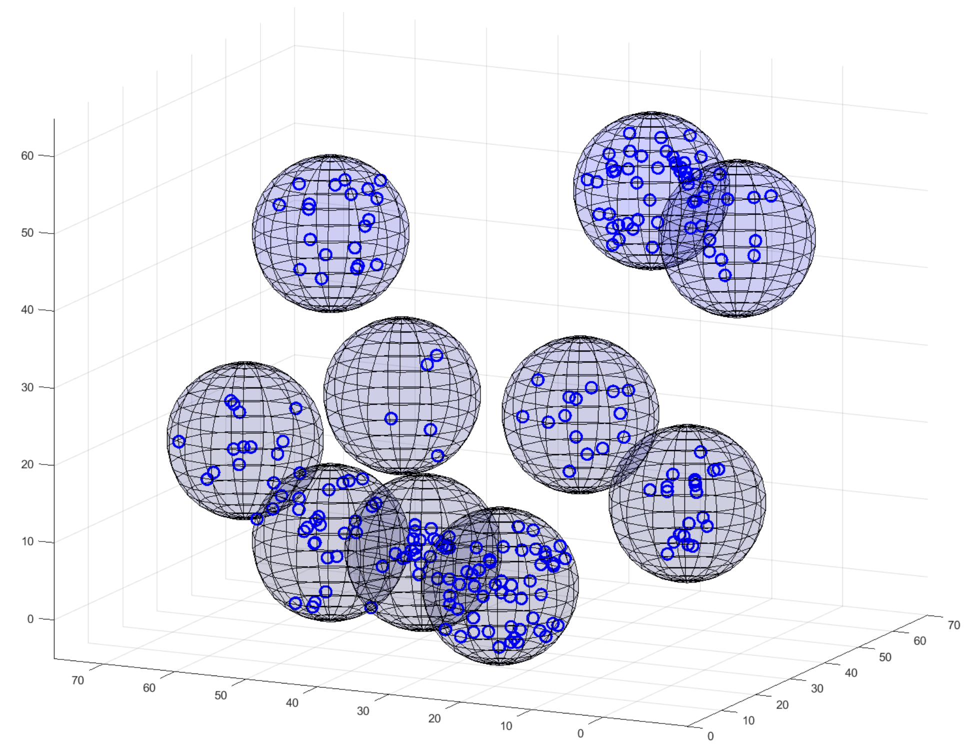


Figure 1: An illustration of the 3D hyperspheres coverage for 230 samples (the blue dots) from a synthetic dataset. All samples in each ball may be represented by its core point: the centre of the ball, which reduces the total number of samples from 230 to just 10.

3.2. The Sketch-and-solve solution

We close this section by explicitly stating our sketch-and-solve approach for speeding-up IVAP. We propose to use AVM in order to efficiently summarise the input data and learn a large-margin separator for our scoring classifier, KSVM. By accelerating the scoring classifier, we will be accelerating the whole IVAP process. We call our sketch-and-solve method *Inductive Venn-Abers Prediction with Enclosing Balls* (IVAP-WEB); that is, IVAPs-WEB are IVAPs that learned the underlying KVSM classifier over summarised data via AVM; more specifically, over core points, each living in its own enclosing ball, as depicted in Figure 1.

Now we are ready to show how the sketch-and-solve paradigm can help IVAP to transcend to the realm of large-scale data.

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. We start our experiments exposition by presenting the data over which our method is validated.

Table 1: Datasets used in validating IVAP-WEB

Dataset	**Training samples**	**Features**
a9a	48,842	123
ijcnn1	141,691	22
w8a	64,700	300

The data presented in Table 1 are large and complex enough to validate our argument that IVAP-WEB indeed provides meaningful acceleration to traditional IVAPs. Each dataset is pre-processed to have zero mean and unit variance.

The dataset a9a contains census information for adult people, including variables such as age, work-class, education, marital status, etc; and the task is to predict whether their yearly income will exceed $50,000. Dataset w8a involves the problem of text categorisation: the data contains different keywords found in web pages and the task is to predict whether the web pages belong to a category or not. Both a9a and w8a were used in Platt (1998a) to show the effectiveness of the SMO algorithm for learning KSVM classifiers. Finally, ijcnn1 is a complex dataset used in a machine learning challenge during IJCNN 2001 (Prokhorov (2001)), and it is well-known for its unbalanced nature *i.e.* about 90% of the instances belongs to the negative class. These three datasets can be found at `https://www.csie.ntu.edu.tw/~cjlin/libsvm/` (last accessed in 7/2020).

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 sketch-and-solve solution IVAP-WEB, described in Section 3, improves the running time of inductive Venn-Abers predictors when KSVM 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 2.3 *i.e.* we use the whole proper training set to train a KSVM classifier; and *IVAP-WEB*, which is the sketch-and-solve approach to speed up IVAPs; that is, we still use the same IVAP framework but the underlying KSVM classifier is trained on a substantially-reduced version of the proper training set, as indicated in Section 3.

Notice that each of the two approaches above needs an *optimisation solver* to train the KSVM classifier. In the IVAP case, we used the previously mentioned LIBSVM solver which, without doubt, has seen tremendous success in efficiently solving the KSVM objective function (see Section 2.1) for large datasets. For IVAP-WEB, we obtain an approximately correct KSVM solution using AVM, which allows us to use enclosing balls to find an SGD solution over a summary of the input points *i.e.* the coreset. Finally, for the sake of completeness, we include computations of using SGD over a uniform random sample of the input points to train a SVM classifier; the idea is to show that coresets are highly desirable over uniform random sampling as they do not arbitrarily lose predictive performance.

4.1. Metrics

Our notion of performance is captured by the below three measures. It is useful to remember at this point that IVAPs produce multi-probability outputs *i.e.* (p_0, p_1), and as anticipated in Section 2.3, we need to adapt these outputs to point-probability type of results. We followed the *minimax* approaches suggested by Vovk *et al.* (Vovk and Petej (2012)). In particular, the authors give two strategies for combining (p_0, p_1) into p depending on the metric to be used [5]; 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 $p := \bar{p} + (p_1 - p_0)\left(\frac{1}{2} - \bar{p}\right)$, where $\bar{p} := (p_0 + p_1)/2$. Finally, notice that for all the metrics considered, small values are preferred over large ones.

- **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.

4.2. Parameters Setting

Regarding AVM, it is very important to have a discussion on its parameters.

4.2.1. Budget Size and Diameter

We mentioned in Section 3.1 that the algorithm is non-parametric with respect to the number of core points. Theoretically, AVM 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 KSVM, 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 2 and 3 show in great detail how the interplay between these two parameters influence different aspects of the final outcome of IVAPs-WEB for the a9a and w8a datasets,

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

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. Note that since the plots for the ijcnn1 dataset are similar to those for a9a, we do not include them for the sake of space.

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 *kernelised* SGD. 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 2(d) and 3(d)), 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 socre). With this analysis in mind, we can proceed to present the rest of the parameters involved in our experiments.

4.2.2. KSVM Parameters

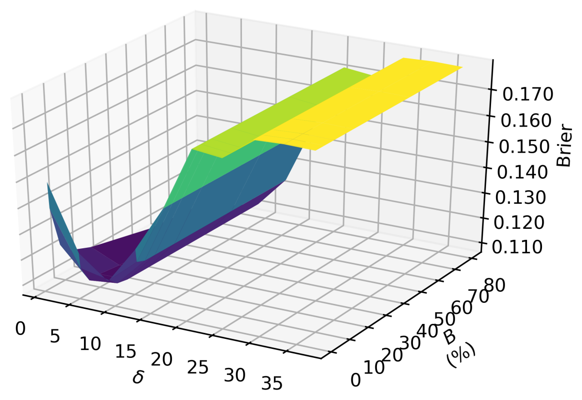
We designed our experiments considering that our underlying scoring classifier, KSVM, is very sensitive to its input parameters. Hence, for the sake of fairness, we decided on the parameters' value as follows: for each method to be tested, and for each dataset, we performed a cross-validation phase over 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 Le et al. (2017): $\lambda \in \{\frac{2^{-4}}{n}, \frac{2^{-3}}{n}, ..., \frac{2^{16}}{n}\}$ and $\delta \in \{1.0, 2.0, ..., 10.0\}$ for AVM [6], 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 2 gives a summary of the values used for each parameter of each method.

4.3. The Evaluation

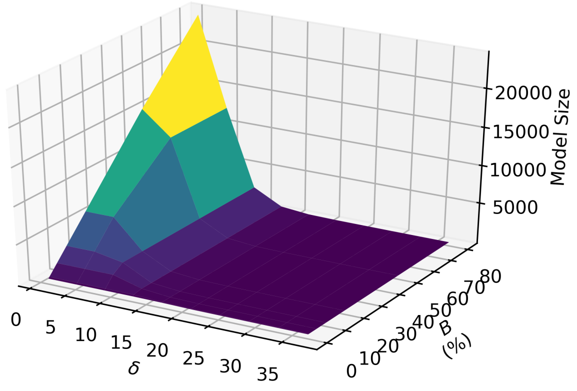
We evaluate IVAP and IVAP-WEB following the below steps. As mentioned previously, we also include a third approach which consists in applying IVAP over a small Uniform Random Sample (URS) of the proper training set. We call this method *IVAP-URS* and the size of the small sample is set to the same size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

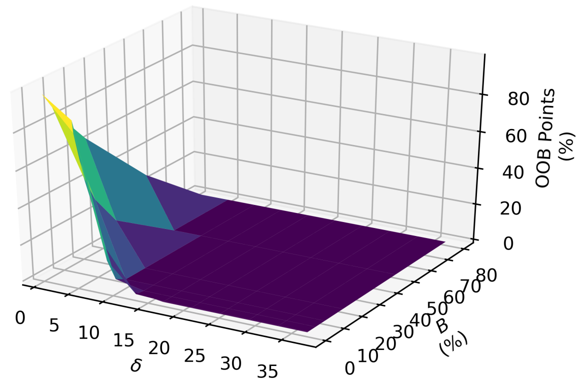
6. LIBSVM always sets λ to 1/2.



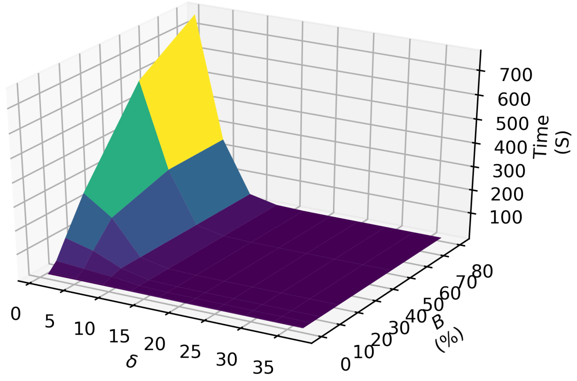
(*a*) Brier score for different diameter and budget-size values.



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



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



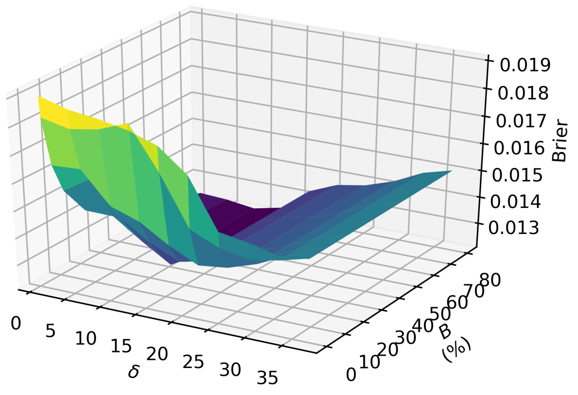
(*d*) Computing time for different diameter and budget-size values.

Figure 2: Interplay between δ and B for the a9a dataset.

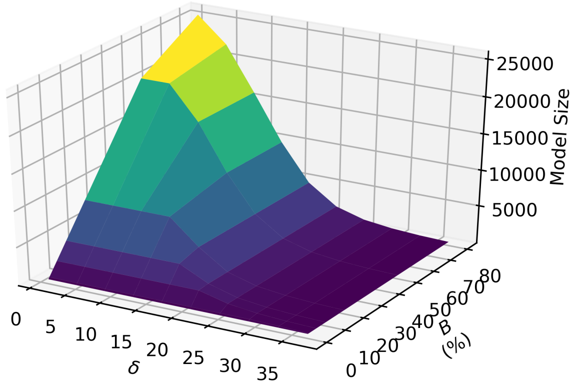
Table 2: Parameters for IVAPs and IVAPs-WEB. IVAPs uses the well-known LIBSVM solver while IVAPs-WEB apply the coreset-based solver AVM.

Parameter	**Algorithm(s)**	**Description**
λ	LIBSVM | AVM	L_2 Regularisation parameter.
γ	LIBSVM | AVM	Kernel coefficient.
δ	AVM	Diameter of balls in input space.
B	AVM	Maximum number of core points (model size) allowed.
C	LIBSVM	Regularisation parameter for dual SVM.

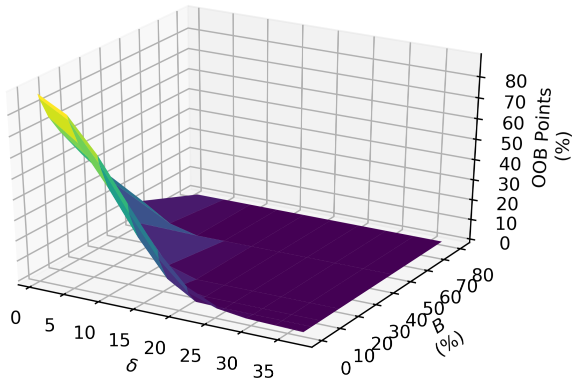
- **Cross-validation:** we then perform 5-fold cross-validation for each of the three approaches. Due to computing limitations, we cross-validate using 1% of the proper training set. The parameter values were taken from the ranges discussed above, in Section 4.2.2.



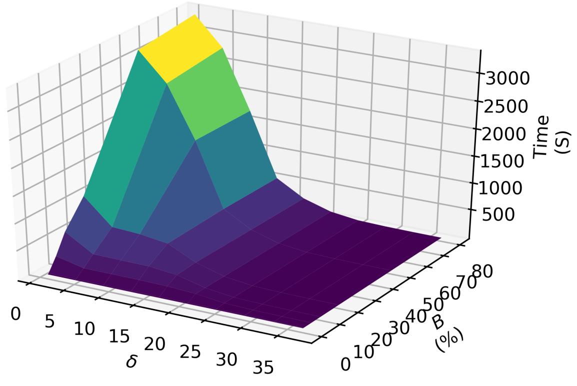
(*a*) Brier score for different diameter and budget-size values.



(*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.

Figure 3: Interplay between δ and B for the w8a dataset.

- **Methods:** we run IVAP, IVAP-WEB and IVAP-URS with the best parameters found.
- **Metrics:** the metrics detailed in Section 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. Regarding the implementations, all the methods and experiments were coded using the Python programming language. AVM's implementation was shared by its own authors (Le et al. (2017)); for LIBSVM and SGD, we used the well-known Scikit-Learn library `https://scikit-learn.org/stable/` (last accessed in 7/2020). For computing IVAPs, Toccaceli's implementation `https://github.com/ptocca/VennABERS` (last accessed in 7/2020) 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.

4.4. Results

In this section, we present and discuss the results obtained in our empirical evaluations. Tables 3, 4 and 5 show the performance of IVAP, IVAP-WEB and IVAP-URS for our three datasets. First, 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 KSVM 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 superior. Interestingly, IVAP-WEB outperforms IVAP in two out of three datasets despite using a very small fraction of the proper training set for learning a KSVM classifier. Hence, 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. 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.

Table 3: Performance comparison for a9a dataset. δ and B are set to 9 and 10% for IVAP-WEB; IVAP-RUS uses a randomly chosen 10% of the proper training set.

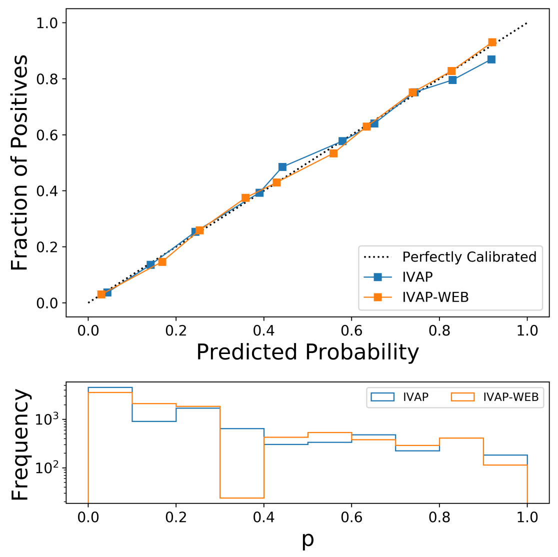
Metric	**IVAP**	**IVAP-WEB**	**IVAP-URS**
Mean Log Error	0.373 ± 0.004	**0.351 ± 0.006**	0.547 ± 0.003
Brier Score	0.117 ± 0.001	**0.112 ± 0.002**	0.180 ± 0.001
Computing Time (s)	586.6 ± 41.9	37.0 ± 6.5	**18.0 ± 0.1**

Table 4: Performance comparison of for ijcnn1 dataset. δ and B are set to 1 and 5% for IVAP-WEB; IVAP-RUS uses a randomly chosen 10% of the proper training set.

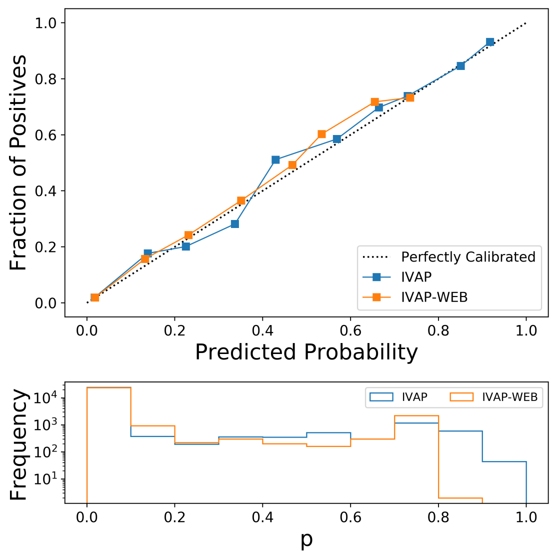
Metric	**IVAP**	**IVAP-WEB**	**IVAP-URS**
Mean Log Error	0.149 ± 0.002	**0.144 ± 0.003**	0.261 ± 0.004
Brier Score	0.042 ± 0.0	**0.038 ± 0.0**	0.070 ± 0.001
Computing Time (s)	557.1 ± 120.8	64.5 ± 5.2	**45.8 ± 0.4**

Table 5: Performance comparison for w8a dataset. δ and B are set to 21 and 10% for IVAP-WEB; IVAP-RUS uses a randomly chosen 10% of the proper training set.

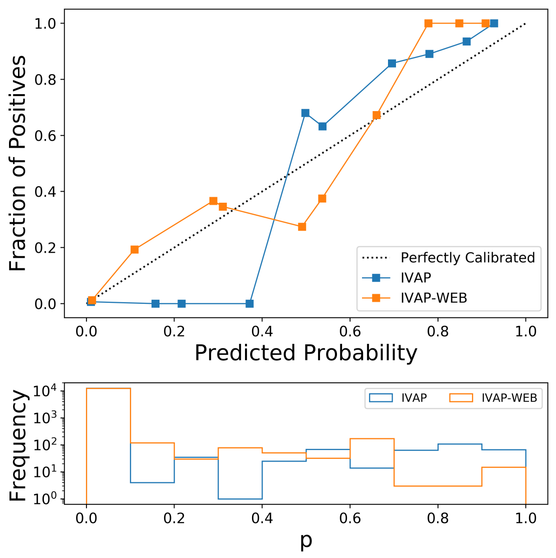
Metric	**IVAP**	**IVAP-WEB**	**IVAP-URS**
Mean Log Error	**0.0465 ± 0.004**	0.0710 ± 0.004	0.135 ± 0.003
Brier Score	**0.0100 ± 0.001**	0.0136 ± 0.001	0.0291 ± 0.0
Computing Time (s)	226.6 ± 19.1	136.3 ± 6.5	**78.6 ± 1.5**



(a) a9a dataset.



(b) ijcnn1 dataset.

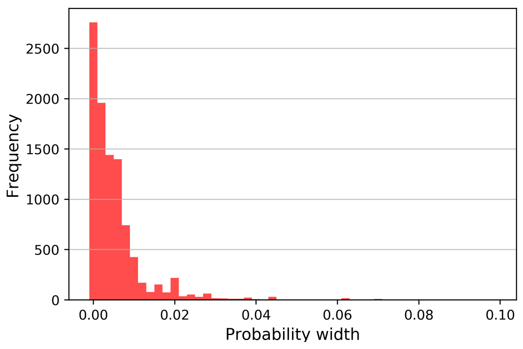


(c) w8a dataset.

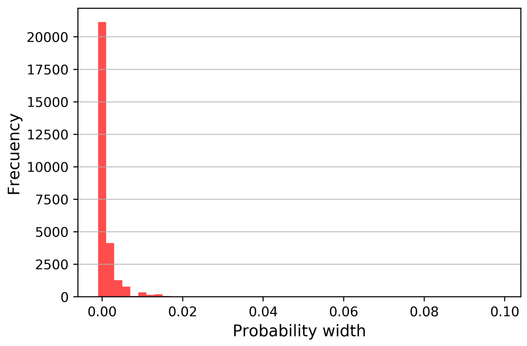
Figure 4: The calibration plots for all 3 datasets. They demonstrate that IVAP-WEB predictions are well-calibrated.

We then evaluate the calibration of IVAP and IVAP-WEB, which can be seen in Figure 4. 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 dataset 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 dataset *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.

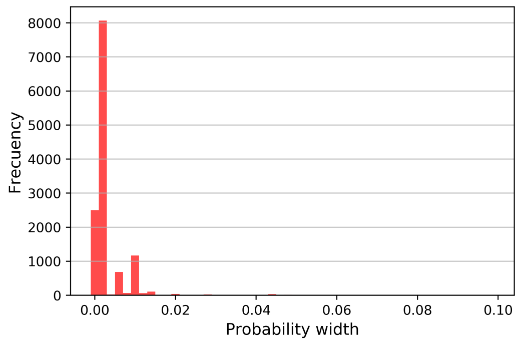
Finally, the efficiency of both IVAP and IVAP-WEB are shown in Figures 5 and 6, respectively. In this context, the notion of efficiency is defined as the difference between p_0 and p_1; that is, the closer they are from each other, the more certain our predictor is and the more efficient the prediction 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.



(a) a9a dataset.

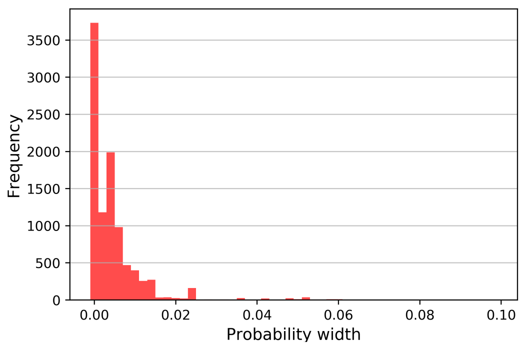


(b) ijcnn1 dataset.

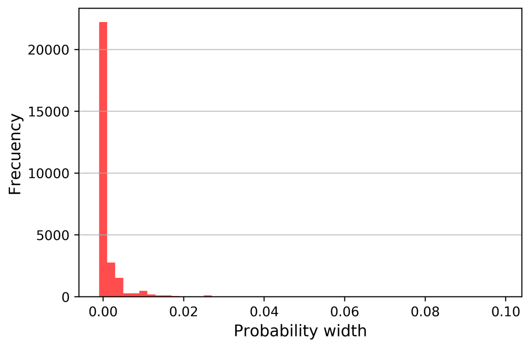


(c) w8a dataset.

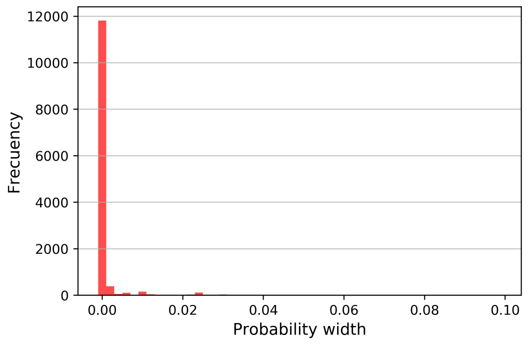
Figure 5: The histogram plots over the probability interval width for IVAP-WEB demonstrates that most intervals are close to zero. Thus, IVAP-WEB retains efficiency.



(a) a9a dataset.



(b) ijcnn1 dataset.



(c) w8a dataset.

Figure 6: The histogram plots over the probability interval width for IVAP demonstrates that most intervals are close to zero. This indicates that most predictions have high certainty.

5. Conclusion

In the era of datasets of unprecedented sizes, scalability tends to be a non-optional trait for machine learning techniques. We concentrated on kernel SVM, a machine learning paradigm that provides tremendous predictive capability at a high computational cost. This cost is exacerbated when we need KSVM to produce probability-type results. To circumvent this, we proposed IVAPs-WEB, IVAPs that train a scoring classifier over a summarised version of the proper training set and that virtually retain the good calibration which characterises IVAPs. The advantage of the sketch-and-solve solution proposed is that it grants great acceleration without modifying KSVM or IVAPs.

Our results indicate that coresets can indeed serve well to the Venn-Abers framework when training a kernel SVM is necessary, and this in turn can prove to be a powerful tool for using IVAP on very large datasets. However, we are also aware that LIBSVM could be at a disadvantage given that the SMO algorithm solves KSVM using the dual formulation and it cannot learn incrementally. Hence, our experiments will keep expanding to potentially include different kind of solvers, especially, new first-order solvers such as SAGA (Defazio et al. (2014)) or SVRG (Harikandeh et al. (2015)).

Acknowledgements

We are thankful to the reviewers for their thorough and constructive comments. The first author is grateful to the Paraguayan Government for funding his research.

References

Miriam Ayer, H Daniel Brunk, George M Ewing, William T Reid, and Edward Silverman. An empirical distribution function for sampling with incomplete information. *The annals of mathematical statistics*, pages 641–647, 1955.

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

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*, pages 201–219, 2018.

Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in neural information processing systems*, pages 1646–1654, 2014.

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

Reza Harikandeh, Mohamed Osama Ahmed, Alim Virani, Mark Schmidt, Jakub Konečnỳ, and Scott Sallinen. Stopwasting my gradients: Practical svrg. In *Advances in Neural Information Processing Systems*, pages 2251–2259, 2015.

Jakub Konecnỳ, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016.

Trung Le, Tu Dinh Nguyen, Vu Nguyen, and Dinh Phung. Approximation vector machines for large-scale online learning. *The Journal of Machine Learning Research*, 18(1):3962–4016, 2017.

Dhruv Mahajan, S Sathiya Keerthi, S Sundararajan, and Léon Bottou. A parallel sgd method with strong convergence. *arXiv preprint arXiv:1311.0636*, 2013.

Alexander Munteanu and Chris Schwiegelshohn. Coresets-methods and history: A theoreticians design pattern for approximation and streaming algorithms. *KI-Künstliche Intelligenz*, 32(1):37–53, 2018.

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

John Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods - Support Vector Learning*. MIT Press, January 1998a. URL https://www.microsoft.com/en-us/research/publication/fast-training-of-support-vector-machines-using-sequential-minimal-optimization/.

John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. *Microsoft Research Technical Report*, 1998b.

Danil Prokhorov. Ijcnn 2001 neural network competition. *Slide presentation in IJCNN*, 1 (97):38, 2001.

Nery Riquelme-Granada, Khuong Nguyen, and Zhiyuan Luo. Coreset-based conformal prediction for large-scale learning. In *Conformal and Probabilistic Prediction and Applications*, pages 142–162, 2019.

Nery 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*, pages 78–89. INSTICC, 2020.

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

Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30, 2011.

Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

Vladimir Vovk and Ivan Petej. Venn-abers predictors. *arXiv preprint arXiv:1211.0025*, 2012.

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

Vladimir Vovk, Ivan Petej, and Valentina Fedorova. Large-scale probabilistic predictors with and without guarantees of validity. In *Advances in Neural Information Processing Systems*, pages 892–900, 2015.

Zhuang Wang, Koby Crammer, and Slobodan Vucetic. Breaking the curse of kernelization: Budgeted stochastic gradient descent for large-scale svm training. *Journal of Machine Learning Research*, 13(Oct):3103–3131, 2012.