

On Coresets for Support Vector Machines^{*}

Murad Tukan^{1,*}

University of Haifa, Computer Science Department, Israel

Cenk Baykal¹

MIT CSAIL, Cambridge, USA

Dan Feldman

University of Haifa, Computer Science Department, Israel

Daniela Rus

MIT CSAIL, Cambridge, USA

Abstract

We present an efficient coreset construction algorithm for large-scale Support Vector Machine (SVM) training in Big Data and streaming applications. A coreset is a small, representative subset of the original data points such that a models trained on the coreset are provably competitive with those trained on the original data set. Since the size of the coreset is generally much smaller than the original set, our preprocess-then-train scheme has potential to lead to significant speedups when training SVM models. We prove lower and upper bounds on the size of the coreset required to obtain small data summaries for the SVM problem. As a corollary, we show that our algorithm can be used to extend the applicability of any off-the-shelf SVM solver to streaming, distributed, and dynamic data settings. We evaluate the performance of our algorithm on real-world and synthetic data sets. Our experimental results reaffirm the

^{*}This research was supported in part by the U.S. National Science Foundation (NSF) under Awards 1723943 and 1526815, Office of Naval Research (ONR) Grant N00014-18-1-2830, Microsoft, and JP Morgan Chase.

^{*}Corresponding author

Email addresses: `muradtuk@gmail.com` (Murad Tukan), `baykal@mit.edu` (Cenk Baykal), `dannyf.post@gmail.com` (Dan Feldman), `rus@mit.edu` (Daniela Rus)

¹These authors contributed equally

favorable theoretical properties of our algorithm and demonstrate its practical effectiveness in accelerating SVM training.

Keywords: **Support vector machines**, Coresets, data compression

1. Introduction

Popular machine learning algorithms are computationally expensive, or worse yet, intractable to train on massive data sets, where the input data set is so large that it may not be possible to process all the data at one time. A natural approach to achieve scalability when faced with Big Data is to first conduct a preprocessing step to summarize the input data points by a significantly smaller, representative set. Off-the-shelf training algorithms can then be run efficiently on this compressed set of data points. The premise of this two-step learning procedure is that the model trained on the compressed set will be provably competitive with the model trained on the original set – as long as the data summary, i.e., the *coreset*, can be generated efficiently and is sufficiently representative.

Coresets are small weighted subsets of the training points such that models trained on the coreset are approximately as good as the ones trained on the original (massive) data set. Coreset constructions were originally introduced in the context of computational geometry [1] and subsequently generalized for applications to other problems, such as logistic regression, neural network compression, and mixture model training [2, 3, 4, 5, 6] (see [7] for a survey).

A popular coreset construction technique – and the one that we leverage in this paper – is to use importance sampling with respect to the points’ *sensitivities*. The sensitivity of each point is defined to be the worst-case relative impact of each data point on the objective function. Points with high sensitivities have a large impact on the objective value and are sampled with correspondingly high probability, and vice-versa. The main challenge in generating small-sized coresets often lies in evaluating the importance of each point in an accurate and computationally-efficient way.

1.1. Our Contributions

In this paper, we propose an efficient coreset construction algorithm to generate compact representations of large data sets to accelerate SVM training. Our approach hinges on bridging the SVM problem with that of k -means clustering. As a corollary to our theoretical analysis, we obtain theoretical justification for the widely reported empirical success of using k -means clustering as a way to generate data summaries for large-scale SVM training. In contrast to prior approaches, our approach is both (i) provably efficient and (ii) naturally extends to streaming or dynamic data settings. Above all, our approach can be used *to enable the applicability of any off-the-shelf SVM solver* – including gradient-based and/or approximate ones, e.g., Pegasos [8], to streaming and distributed data settings by exploiting the *composability* and *reducibility* properties of coresets [7].

In particular, this paper contributes the following:

1. A coreset construction algorithm for accelerating SVM training based on an efficient importance sampling scheme.
2. An analysis proving lower bounds on the number of samples required by any coreset construction algorithm to approximate the input data set.
3. Theoretical guarantees on the efficiency and accuracy of our coreset construction algorithm.
4. Evaluations on synthetic and real-world data sets that demonstrate the effectiveness of our algorithm in both streaming and offline settings.

2. Related Work

Training SVMs requires $\mathcal{O}(n^3)$ time and $\mathcal{O}(n^2)$ space in the offline setting where n is the number of training points. Towards the goal of accelerating SVM training in the offline setting, [9, 10] introduced the Core Vector Machine (CVM) and Ball Vector Machine (BVM) algorithms, which are based on reformulating the SVM problem as the Minimum Enclosing Ball (MEB) problem and

Enclosing Ball (EB) problem, respectively, and by leveraging existing coresets constructions for each; see [11]. However, CVM’s accuracy and convergence properties have been noted to be at times inferior relative to those of existing SVM implementations [12]; moreover, unlike the algorithm presented in this paper, neither the CVM, nor the BVM algorithm extends naturally to streaming or dynamic settings where data points are continuously inserted or deleted. Similar geometric approaches, including extensions of the MEB formulation, those based on convex hulls and extreme points, among others, were investigated by [13, 14, 15, 16, 17, 18]. Another class of related work includes the use of canonical optimization algorithms such as the Frank-Wolfe algorithm [19], Gilbert’s algorithm [19, 20], and a primal-dual approach combined with Stochastic Gradient Descent (SGD) [21].

SGD-based approaches, such as Pegasos [8], have been a popular tool of choice in approximately-optimal SVM training. Pegasos is a stochastic sub-gradient algorithm for obtaining a $(1 + \varepsilon)$ -approximate solution to the SVM problem in $\tilde{O}(dn\lambda/\varepsilon)$ time for a linear kernel, where λ is the regularization parameter and d is the dimensionality of the input data points. In contrast to our method, these approaches and their corresponding theoretical guarantees do not feasibly extend to dynamic data sets and/or streaming settings. In particular, gradient-based approaches cannot be trivially extended to streaming settings since the arrival of each input point in the stream results in a change of the gradient.

There has been prior work in streaming algorithms for SVMs, such as those of [13, 15, 22, 18]. However, these works generally suffer from poor practical performance in comparison to that of approximately optimal SVM algorithms in the offline (batch) setting, high difficulty of implementation and application to practical settings, and/or lack of strong theoretical guarantees. Unlike the algorithms of prior work, our method is simultaneously simple-to-implement, exhibits theoretical guarantees, and naturally extends to streaming and dynamic data settings, where the input data set is so large that it may not be possible to store or process all the data at one time.

3. Problem Definition

Let $P = \{(x, y) : x \in \mathbb{R}^d \times 1, y \in \{\pm 1\}\}$ denote a set of n input points. Note that for each point $p = (x, y) \in P$, the last entry $x_{d+1} = 1$ of x accounts for the bias term embedding into the feature space². To present our results with full generality, we consider the setting where the input points P may have weights associated with them. Hence, given P and a weight function $u : P \rightarrow \mathbb{R}_{\geq 0}$, we let $\mathcal{P} = (P, u)$ denote the weighted set with respect to P and u . The canonical unweighted case can be represented by the weight function that assigns a uniform weight of 1 to each point, i.e., $u(p) = 1$ for every point $p \in P$. For every $T \subseteq P$, let $U(T) = \sum_{p \in T} u(p)$. We consider the scenario where n is much larger than the dimension of the data points, i.e., $n \gg d$.

For a normal to a separating hyperplane $w \in \mathbb{R}^{d+1}$, let $w_{1:d}$ denote vector which contains the first d entries of w . The last entry of w (w_{d+1}) encodes the bias term $b \in \mathbb{R}$. Under this setting, the hinge loss of any point $p = (x, y) \in P$ with respect to a normal to a separating hyperplane, $w \in \mathbb{R}^{d+1}$, is defined as $h(p, w) = [1 - y\langle x, w \rangle]_+$, where $[\cdot]_+ = \max\{0, \cdot\}$. As a prelude to our subsequent analysis of sensitivity-based sampling, we quantify the contribution of each point $p = (x, y) \in P$ to the SVM objective function as

$$f_\lambda(p, w) = \frac{1}{2U(P)} \|w_{1:d}\|_2^2 + \lambda h(p, w), \quad (1)$$

where $\lambda \in [0, 1]$ is the SVM regularization parameter, and $h(p, w) = [1 - y\langle x, w \rangle]_+$ is the hinge loss with respect to the query $w \in \mathbb{R}^{d+1}$ and point $p = (x, y)$. Putting it all together, we formalize the λ -regularized SVM problem as follows.

Definition 1 (λ -regularized SVM Problem). *For a given weighted set of points $\mathcal{P} = (P, u)$ and a regularization parameter $\lambda \in [0, 1]$, the λ -regularized SVM problem with respect to $\mathcal{P}$ is given by*

$$\min_{w \in \mathbb{R}^{d+1}} F_\lambda(\mathcal{P}, w),$$

²We perform this embedding for ease of presentation later on in our analysis.

where

$$F_\lambda(\mathcal{P}, w) = \sum_{p \in \mathcal{P}} u(p) f(p, w). \quad (2)$$

We let w^* denote the optimal solution to the SVM problem with respect to $\mathcal{P}$, i.e., $w^* \in \operatorname{argmin}_{w \in \mathbb{R}^{d+1}} F_\lambda(\mathcal{P}, w)$. A solution $\hat{w} \in \mathbb{R}^{d+1}$ is an ξ -approximation to the SVM problem if $F_\lambda(\mathcal{P}, \hat{w}) \leq F_\lambda(\mathcal{P}, w^*) + \xi$. Next, we formalize the *coreset guarantee* that we will strive for when constructing our data summaries.

Coresets. A coreset is a compact representation of the full data set that provably approximates the SVM cost function (2) for every query $w \in \mathbb{R}^{d+1}$ – including that of the optimal solution w^* . We formalize this notion below for the SVM problem with objective function $F_\lambda(\cdot)$ as in (2) below.

Definition 2 (ε -coreset). *Let $\varepsilon \in (0, 1)$ and let $\mathcal{P} = (P, u)$ be the weighted set of training points as before. A weighted subset $\mathcal{S} = (S, v)$, where $S \subset P$ and $v : S \rightarrow \mathbb{R}_{\geq 0}$ is an ε -coreset for $\mathcal{P}$ if*

$$\forall w \in \mathbb{R}^{d+1} \quad |F_\lambda(\mathcal{P}, w) - F_\lambda(\mathcal{S}, w)| \leq \varepsilon F_\lambda(\mathcal{P}, w). \quad (3)$$

This strong guarantee implies that the models trained on the coreset $\mathcal{S}$ with any off-the-shelf SVM solver will be approximately (and provably) as good as the optimal solution w^* obtained by training on the entire data set $\mathcal{P}$. This also implies that, if the size of the coreset is provably small, e.g., logarithmic in n (see Sec. 5), then an approximately optimal solution can be obtained much more quickly by training on $\mathcal{S}$ rather than $\mathcal{P}$, leading to computational gains in practice for both offline and streaming data settings (see Sec. 6).

The difficulty in constructing coresets lies in constructing them (i) *efficiently*, so that the preprocess-then-train pipeline takes less time than training on the full data set and (ii) *accurately*, so that important data points – i.e., those that are imperative to obtaining accurate models – are not left out of the coreset, and redundant points are eliminated so that the coreset size is small. In the following sections, we introduce and analyze our coreset algorithm for the SVM problem.

4. Method

Our coresets construction scheme is based on the unified framework of [4, 5] and is shown in Alg. 1. The crux of our algorithm lies in generating the importance sampling distribution via efficiently computable upper bounds (proved in Sec. 5) on the importance of each point (Lines 1-10). Sufficiently many points are then sampled from this distribution and each point is given a weight that is inversely proportional to its sample probability (Lines 11-12). The number of points required to generate an ε -coreset with probability at least $1 - \delta$ is a function of the desired accuracy ε , failure probability δ , and complexity of the data set (t from Theorem 7). Under mild assumptions on the problem at hand (see Sec. Appendix A.4), the required sample size is polylogarithmic in n .

Our algorithm is an importance sampling procedure that first generates a judicious sampling distribution based on the structure of the input points and samples sufficiently many points from the original data set. The resulting weighted set of points $\mathcal{S} = (S, v)$, serves as an unbiased estimator for $F_\lambda(\mathcal{P}, w)$ for any query $w \in \mathbb{R}^{d+1}$, i.e., $\mathbb{E}[F_\lambda(\mathcal{S}, w)] = F_\lambda(\mathcal{P}, w)$. Although sampling points uniformly with appropriate weights can also generate such an unbiased estimator, it turns out that the variance of this estimation is minimized if the points are sampled according to the distribution defined by the ratio between each point's sensitivity and the sum of sensitivities, i.e., $\gamma(p)/t$ on Line 12 [23].

4.1. Computational Complexity

Coresets are intended to provide efficient and provable approximations to the optimal SVM solution. However, the very first line of our algorithm entails computing an (approximately) optimal solution to the SVM problem. This seemingly eerie phenomenon is explained by the merge-and-reduce technique [24] that ensures that our coreset algorithm is only run against small partitions of the original data set [3, 24, 25]. The merge-and-reduce approach leverages the fact that coresets are composable and reduces the coreset construction problem for a (large) set of n points into the problem of computing coresets for $\frac{n}{2^{|S|}}$

Algorithm 1: CORESET(P, u, λ, ξ, k, m)

Input : A set of training points $P \subseteq \mathbb{R}^{d+1} \times \{-1, 1\}$ containing n points, weight function $u : P \rightarrow \mathbb{R}_{\geq 0}$, a regularization parameter $\lambda \in [0, 1]$, an approximation factor $\xi > 0$, a positive integer k , a sample size m

Output : An weighted set (S, v) which satisfies Theorem [7](#)

```
1  $\tilde{w} \leftarrow$  An  $\xi$ -approximation for the optimal SVM of  $(P, u)$ ;  
2  $\widetilde{opt}_\xi \leftarrow F_\lambda(\mathcal{P}, \tilde{w}) - \xi$ ;  
3 for  $y \in \{-, +\}$  do  
4    $P_y \leftarrow$  all the points in  $P$  that are associated with the label  $y$ ;  
5    $(c_y^{(i)}, P_y^{(i)})_{i=1}^k \leftarrow \text{K-MEANS++}(\mathcal{P}, k)$ ;  
6   for every  $i \in [k]$  do  
7      $\alpha_y^{(i)} \leftarrow \frac{U(P \setminus P_y^{(i)})}{2\lambda U(P)U(P_y^{(i)})}$ ;  
8     for every  $p = (x, y) \in P_y^{(i)}$  do  
9        $p_\Delta \leftarrow c_y^{(i)} - yx$ ;  
10       $\gamma(p) \leftarrow$   
           $\frac{u(p)}{U(P_y^{(i)})} + \lambda u(p)^{\frac{9}{2}} \max \left\{ \frac{4}{9} \alpha_y^{(i)}, \sqrt{4 \left( \alpha_y^{(i)} \right)^2 + \frac{2 \|p_\Delta\|_2^2}{9 \widetilde{opt}_\xi}} - 2 \alpha_y^{(i)} \right\}$ ;  
11  $t \leftarrow \sum_{p \in P} \gamma(p)$ ;  
12  $(S, v) \leftarrow m$  weighted samples from  $\mathcal{P} = (P, u)$  where each point  $p \in P$  is  
    sampled with probability  $q(p) = \frac{\gamma(p)}{t}$  and, if sampled, has weight  
     $v(p) = \frac{u(p)}{mq(p)}$ ;  
13 return  $(S, v)$ 
```

points, where $2|S|$ is the minimum size of input set that can be reduced to half using Algorithm [1](#) [3](#). Assuming that the sufficient conditions for obtaining polylogarithmic size coresets implied by Theorem [7](#) hold, the overall time required is approximately linear in n .

5. Analysis

In this section, we analyze the sample-efficiency and computational complexity of our algorithm. The outline of this section is as follows: we first formalize the importance (i.e., *sensitivity*) of each point and summarize the necessary conditions for the existence of small coresets. We then present the negative result that, in general, sublinear coresets do not exist for *every* data set (Lem. 4). Despite this, we show that we can obtain accurate approximations for the sensitivity of each point via an approximate k -means clustering (Lems. 5 and 6), and present non-vacuous, data-dependent bounds on the sample complexity (Thm. 7).

5.1. Preliminaries

We will henceforth state all of our results with respect to the weighted set of training points $\mathcal{P} = (P, u)$, $\lambda \in [0, 1]$, and SVM cost function F_λ (as in Sec. 3). The definition below rigorously quantifies the *relative contribution* of each point.

Definition 3 (Sensitivity 3). *The sensitivity of each point $p \in P$ is given by*

$$s(p) = \sup_w \frac{u(p)f_\lambda(p, w)}{F_\lambda(\mathcal{P}, w)}. \quad (4)$$

Note that in practice, exact computation of the sensitivity is intractable, so we usually settle for (sharp) upper bounds on the sensitivity $\gamma(p) \geq s(p)$ (e.g., as in Alg. 1). Sensitivity-based importance sampling then boils down to normalizing the sensitivities by the normalization constant – to obtain an importance sampling distribution – which in this case is the *sum of sensitivities* $t = \sum_{p \in P} s(p)$. It turns out that the required size of the coreset is at least linear in t 3, which implies that one immediate necessary condition for sublinear coresets is $t \in o(n)$.

5.2. Lower bound for Sensitivity

The next lemma shows that a sublinear-sized coreset cannot be constructed for *every* SVM problem instance. The proof of this result is based on demonstrating

a hard point set for which the sum of sensitivities is $\Omega(n\lambda)$, ignoring d factors, which implies that sensitivity-based importance sampling roughly boils down to uniform sampling for this data set. This in turn implies that if the regularization parameter is too large, e.g., $\lambda = \theta(1)$, and if $d \ll n$ (as in Big Data applications) then the required number of samples for property [3](#) to hold is $\Omega(n)$.

Lemma 4. *For an even integer $d \geq 2$, there exists a set of weighted points $\mathcal{P} = (P, u)$ such that*

$$s(p) \geq \frac{n\lambda + d^2}{n(\lambda + d^2)} \quad \forall p \in P \quad \text{and} \quad \sum_{p \in P} s(p) \geq \frac{n\lambda + d^2}{(\lambda + d^2)}.$$

We next provide upper bounds on the sensitivity of each data point with respect to the complexity of the input data. Despite the non-existence results established above, our upper bounds shed light into the class of problems for which small-sized coresets are ensured to exist.

5.3. Sensitivity Upper Bound

In this subsection we present sharp, data-dependent upper bounds on the sensitivity of each point. Our approach is based on an approximate solution to the k -means clustering problem and to the SVM problem itself (as in Alg. [1](#)). To this end, we will henceforth let k be a positive integer, $\xi \in [0, F_\lambda(\mathcal{P}, w^*)]$ be the error of the (coarse) SVM approximation, and let $(c_y^{(i)}, P_y^{(i)})$, $\alpha_y^{(i)}$ and p_Δ for every $y \in \{+, -\}$, $i \in [k]$ and $p \in P$ as in Lines [4-9](#) of Algorithm [1](#).

Lemma 5. *Let k be a positive integer, $\xi \in [0, F_\lambda(\mathcal{P}, w^*)]$, and let $\mathcal{P} = (P, u)$ be a weighted set. Then for every $i \in [k]$, $y \in \{+, -\}$ and $p \in P_y^{(i)}$,*

$$s(p) \leq \frac{u(p)}{U(P_y^{(i)})} + \lambda u(p) \frac{9}{2} \max \left\{ \frac{4}{9} \alpha_y^{(i)}, \sqrt{4 \left(\alpha_y^{(i)} \right)^2 + \frac{2 \|p_\Delta\|_2^2}{9 \widetilde{\text{opt}}_\xi}} - 2 \alpha_y^{(i)} \right\} = \gamma(p).$$

Lemma 6. *In the context of Lemma [5](#), the sum of sensitivities is bounded by*

$$\sum_{p \in P} s(p) \leq t = 4k + \sum_{i=1}^k \frac{3\lambda \text{Var}_+^{(i)}}{\sqrt{2\widetilde{\text{opt}}_\xi}} + \frac{3\lambda \text{Var}_-^{(i)}}{\sqrt{2\widetilde{\text{opt}}_\xi}},$$

where $\text{Var}_y^{(i)} = \sum_{p \in P_y^{(i)}} u(p) \|p_\Delta\|_2$ for all $i \in [k]$ and $y \in \{+, -\}$.

Theorem 7. For any $\varepsilon \in (0, 1/2)$, $\delta \in (0, 1)$, let m be an integer satisfying

$$m \in \Omega \left(\frac{t}{\varepsilon^2} (d \log t + \log(1/\delta)) \right),$$

where t is as in Lem. [6](#). Invoking CORESET with the inputs defined in this context yields a ε -coreset $\mathcal{S} = (S, v)$ with probability at least $1 - \delta$ in $\mathcal{O}(nd + T)$ time, where T represents the computational complexity of obtaining an ξ -approximated solution to SVM and applying k -means++ on P_+ and P_- .

Sufficient Conditions and the effect of k means on our sensitivity.. Theorem [7](#) immediately implies that, for reasonable ε and δ , coresets of poly-logarithmic (in n) size can be obtained if $d = \mathcal{O}(\text{polylog}(n))$, which is usually the case in our target Big Data applications, and if $\sum_{i=1}^k \frac{3\lambda \text{Var}_+^{(i)}}{\sqrt{2\text{opt}_\xi}} + \frac{3\lambda \text{Var}_-^{(i)}}{\sqrt{2\text{opt}_\xi}} = \mathcal{O}(\text{polylog}(n))$.

Despite the fact that any k -partitioning of the data can be applied instead of k -means for achieving upper bound on the sensitivities of the points, its important to note that k -means actually acts as a trade-off mechanism between the *raw contribution* and the *actual contribution* (the weight term and the max term from Lemma [5](#), respectively). Choosing the best k can be done via binary search over the values of k that minimize the sensitivity. We refer the reader to literature on the Silhouette and Elbow methods [\[26\]](#) as ways to pick the optimal k .

6. Results

In this section, we present experimental results that demonstrate and compare the effectiveness of our algorithm on a variety of synthetic and real-world data sets in offline and streaming data settings [\[27\]](#). Our empirical evaluations demonstrate the practicality and wide-spread effectiveness of our approach: our algorithm consistently generated more compact and representative data summaries, and yet incurred a negligible increase in computational complexity when compared to uniform sampling.

Dataset Measurements	HTRU	Credit	Pathol.	Skin	Cod	W1
Number of data-points (n)	17,898	30,000	1,000	245,057	488,565	49,749
Sum of Sensitivities (t)	475.8	1,013.0	77.6	271.5	2,889.2	24,231.6
t/n (Percentage)	2.7%	3.4%	7.7%	0.1%	0.6%	51.3%

Table 1: The number of input points and measurements of the total sensitivity computed empirically for each data set in the offline setting. The sum of sensitivities is significantly less than n for virtually all of the data sets, which, by Thm. 7 ensures the sample-efficiency of our approach on the evaluated scenarios.

Evaluation. We considered 6 real-world data sets of varying size and complexity as depicted in Table 1. For each data set of size n , we selected a set of $M = 15$ geometrically-spaced subsample sizes $m_1, \dots, m_M \subset [\log n, n^{4/5}]$. For each sample size m , we ran each algorithm (Alg. 1 or uniform sampling) to construct a subset $\mathcal{S} = (S, v)$ of size m . We then trained the SVM model as per usual on this subset to obtain an optimal solution with respect to the coreset $\mathcal{S}$, i.e., $w_{\mathcal{S}}^* = \operatorname{argmin}_w F_{\lambda}(\mathcal{S}, w)$. We then computed the relative error incurred by the solution computed on the coreset ($w_{\mathcal{S}}^*$) with respect to the ground-truth optimal solution computed on the entire data set (w^*): $|F_{\lambda}(P, w_{\mathcal{S}}^*) - F_{\lambda}(P, w^*)| / F_{\lambda}(P, w^*)$. The results were averaged across 100 trials.

Figures 1 and 2 depict the results of our comparisons against uniform sampling in the offline setting. In Fig. 1, we see that the coresets generated by our algorithm are much more representative and compact than the ones constructed by uniform sampling: across all data sets and sample sizes, training on our coreset yields significantly better solutions to SVM problem when compared to those generated by training on a uniform sample. For certain data sets, such as HTRU, Pathological, and W1, this relative improvement over uniform sampling is at least an order of magnitude better, especially for small sample sizes. Fig. 1 also shows that, as a consequence of a more informed sampling scheme, the variance of each model’s performance trained on our coreset is much lower than that of uniform sampling for all data sets.

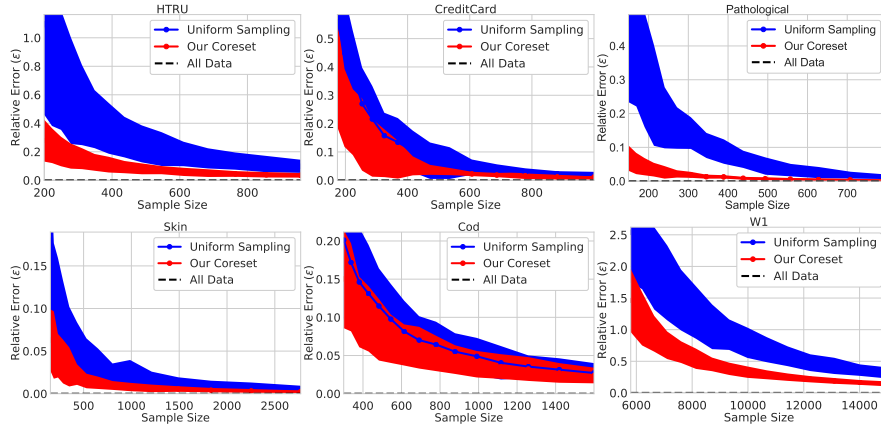


Figure 1: The relative error of query evaluations with respect uniform and coreset subsamples for the 6 data sets in the offline setting. Shaded region corresponds to values within one standard deviation of the mean.

Fig. 2 shows the total computational time required for constructing the sub-sample (i.e., coreset) $\mathcal{S}$ and training the SVM on the subset $\mathcal{S}$ to obtain $w_{\mathcal{S}}^*$. We observe that our approach takes significantly less time than training on the original model when considering non-trivial data sets (i.e., $n \geq 18,000$), and underscores the efficiency of our method: we incur a negligible cost in the overall SVM training time due to a more involved coreset construction procedure, but benefit heavily in terms of the accuracy of the models generated (Fig. 1).

Next, we evaluate our approach in the streaming setting, where data points arrive one-by-one and the entire data set cannot be kept in memory, for the same 6 data sets. The results of the streaming setting are shown in Fig. 3. Figs. 3 portray a similar trend as the one we observed in our offline evaluations: our approach significantly outperforms uniform sampling for all of the evaluated data sets and sample sizes, with negligible computational overhead.

In sum, our empirical evaluations demonstrate the practical efficiency of our algorithm and reaffirm the favorable theoretical guarantees of our approach: the additional computational complexity of constructing the coreset is negligible relative to that of uniform sampling, and the entire preprocess-then-train pipeline is significantly more efficient than training on the original massive data set.

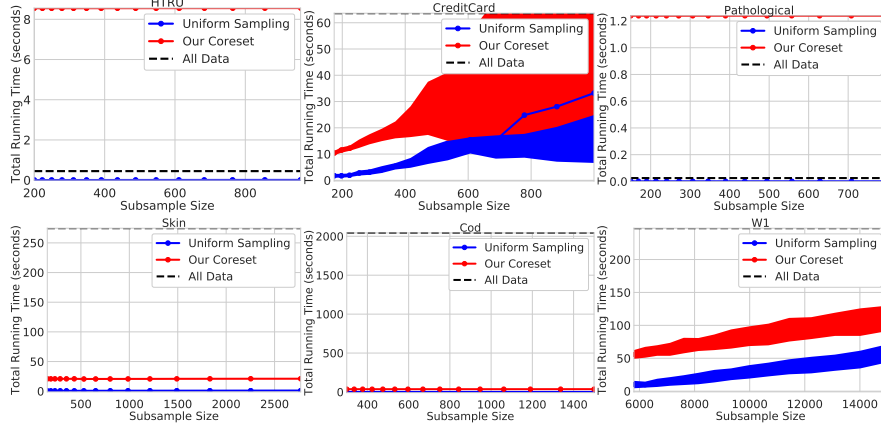


Figure 2: The *total* computational cost of constructing a coreset and training the SVM model on the coreset, plotted as a function of the size of the coreset.

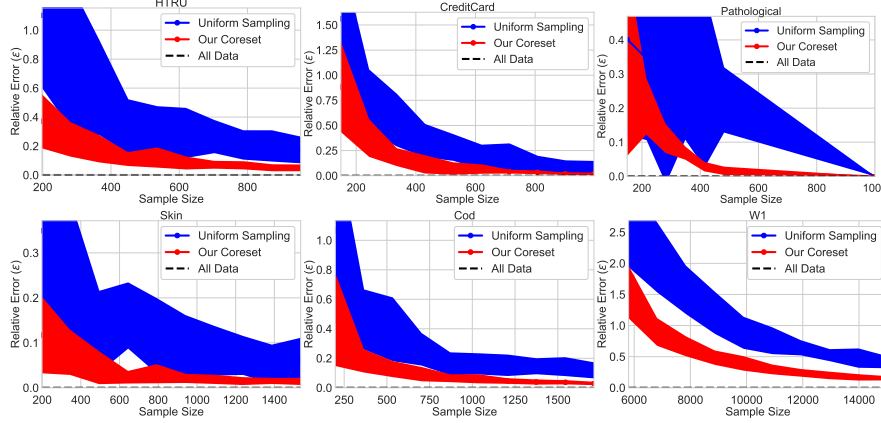


Figure 3: The relative error of query evaluations with respect uniform and coreset subsamples for the 6 data sets in the streaming setting. The figure shows that our method tends to fare even better in the streaming setting (cf. Fig. 1).

7. Conclusion

We presented an efficient coreset construction algorithm for generating compact representations of the input data points that are provably competitive with the original data set in training Support Vector Machine models. Unlike prior approaches, our method and its theoretical guarantees naturally extend to streaming settings and scenarios involving dynamic data sets, where points are

continuously inserted and deleted. We established instance-dependent bounds on the number of samples required to obtain accurate approximations to the SVM problem as a function of input data complexity and established dataset dependent conditions for the existence of compact representations. Our experimental results on real-world data sets validate our theoretical results and demonstrate the practical efficacy of our approach in speeding up SVM training. We conjecture that our coreset construction can be extended to accelerate SVM training for other classes of kernels and can be applied to a variety of Big Data scenarios.

References

- [1] P. K. Agarwal, S. Har-Peled, K. R. Varadarajan, Geometric approximation via coresets, *Combinatorial and computational geometry* 52 (2005) 1–30.
- [2] C. Baykal, L. Liebenwein, I. Gilitschenski, D. Feldman, D. Rus, Data-dependent coresets for compressing neural networks with applications to generalization bounds, *arXiv preprint arXiv:1804.05345*.
- [3] V. Braverman, D. Feldman, H. Lang, New frameworks for offline and streaming coreset constructions, *arXiv preprint arXiv:1612.00889*.
- [4] D. Feldman, M. 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.
- [5] M. Langberg, L. J. Schulman, Universal ε -approximators for integrals, in: *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, SIAM, 2010, pp. 598–607.
- [6] L. Liebenwein, C. Baykal, H. Lang, D. Feldman, D. Rus, Provable filter pruning for efficient neural networks, *arXiv preprint arXiv:1911.07412*.
- [7] D. Feldman, Core-sets: An updated survey, *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* (2019) e1335.

- [8] S. Shalev-Shwartz, Y. Singer, N. Srebro, A. Cotter, Pegasos: Primal estimated sub-gradient solver for svm, *Mathematical programming* 127 (1) (2011) 3–30.
- [9] I. W. Tsang, A. Kocsor, J. T. Kwok, Simpler core vector machines with enclosing balls, in: *Proceedings of the 24th international conference on Machine learning*, ACM, 2007, pp. 911–918.
- [10] I. W. Tsang, J. T. Kwok, P.-M. Cheung, Core vector machines: Fast svm training on very large data sets, *Journal of Machine Learning Research* 6 (Apr) (2005) 363–392.
- [11] M. Badoiu, K. 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.
- [12] G. Loosli, S. Canu, Comments on the “Core Vector Machines: Fast SVM Training on Very Large Data Sets”, *Journal of Machine Learning Research* 8 (Feb) (2007) 291–301.
- [13] P. K. Agarwal, R. Sharathkumar, Streaming algorithms for extent problems in high dimensions, in: *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, Society for Industrial and Applied Mathematics, 2010, pp. 1481–1489.
- [14] B. Gärtner, M. Jaggi, Coresets for polytope distance, in: *Proceedings of the twenty-fifth annual symposium on Computational geometry*, ACM, 2009, pp. 33–42.
- [15] S. Har-Peled, D. Roth, D. Zimak, Maximum margin coresets for active and noise tolerant learning., in: *IJCAI*, 2007, pp. 836–841.
- [16] T. Joachims, Training linear svms in linear time, in: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2006, pp. 217–226.

- [17] M. Nandan, P. P. Khargonekar, S. S. Talathi, Fast svm training using approximate extreme points., *Journal of Machine Learning Research* 15 (1) (2014) 59–98.
- [18] P. Rai, H. Daumé III, S. Venkatasubramanian, Streamed learning: one-pass svms, *arXiv preprint arXiv:0908.0572*.
- [19] K. L. Clarkson, Coresets, sparse greedy approximation, and the frank-wolfe algorithm, *ACM Transactions on Algorithms (TALG)* 6 (4) (2010) 63.
- [20] K. L. Clarkson, E. Hazan, D. P. Woodruff, Sublinear optimization for machine learning, *Journal of the ACM (JACM)* 59 (5) (2012) 23.
- [21] E. Hazan, T. Koren, N. Srebro, Beating sgd: Learning svms in sublinear time, in: *Advances in Neural Information Processing Systems*, 2011, pp. 1233–1241.
- [22] V. Nathan, S. Raghvendra, Accurate streaming support vector machines, *arXiv preprint arXiv:1412.2485*.
- [23] O. Bachem, M. Lucic, A. Krause, Practical coreset constructions for machine learning, *arXiv preprint arXiv:1703.06476*.
- [24] S. Har-Peled, S. Mazumdar, On coresets for k-means and k-median clustering, in: *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, ACM, 2004, pp. 291–300.
- [25] M. Lucic, M. Faulkner, A. Krause, D. Feldman, Training mixture models at scale via coresets, *arXiv preprint arXiv:1703.08110*.
- [26] T. M. Kodinariya, P. R. Makwana, Review on determining number of cluster in k-means clustering, *International Journal* 1 (6) (2013) 90–95.
- [27] M. Lichman, [UCI machine learning repository](http://archive.ics.uci.edu/ml) (2013).
URL <http://archive.ics.uci.edu/ml>

- [28] J. Yang, Y.-L. Chow, C. Ré, M. W. Mahoney, Weighted sgd for ℓ_p regression with randomized preconditioning, arXiv preprint arXiv:1502.03571.
- [29] V. N. Vapnik, V. Vapnik, Statistical learning theory, Vol. 1, Wiley New York, 1998.
- [30] Y. Li, P. M. Long, A. Srinivasan, Improved bounds on the sample complexity of learning, Journal of Computer and System Sciences 62 (3) (2001) 516–527.
- [31] D. Arthur, S. 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.

Appendix A. Proofs of the Analytical Results in Section 5

This section includes the full proofs of the technical results given in Sec. 5

Appendix A.1. Proof of Lemma 4

Lemma 4. *For an even integer $d \geq 2$, there exists a set of weighted points $\mathcal{P} = (P, u)$ such that*

$$s(p) \geq \frac{n\lambda + d^2}{n(\lambda + d^2)} \quad \forall p \in P \quad \text{and} \quad \sum_{p \in P} s(p) \geq \frac{n\lambda + d^2}{(\lambda + d^2)}.$$

Proof. Following [28], let $n = \binom{d}{d/2}$ and let $\mathcal{P} = (P, u)$, where $P \subseteq \mathbb{R}^{d+1} \times \{\pm 1\}$ be set of n labeled points, and $u : P \rightarrow 1$. For every $p = (x, y) \in P$, where $x \in \mathbb{R}^d \times \{1\}$ and $y \in \{\pm 1\}$, among the first d entries of x , exactly $\frac{d}{2}$ entries are equivalent to

$$y\sqrt{\frac{2}{d}},$$

where the remaining $\frac{d}{2}$ entries among the first d are set to 0. Hence, for our proof to hold, we assume that P contains all such combinations and at least one point of each label. For every $p = (x, y) \in P$, define the set of non-zero entries of p as the set

$$B_p = \{i \in [d+1] : x_i \neq 0\}.$$

Put $p \in P$ and note that for bounding the sensitivity of point p , consider w with entries defined as

$$\forall i \in [d+1] \quad w_i = \begin{cases} 0 & \text{if } i \in B_p, \\ \frac{1}{\sqrt{\frac{2}{d}}} & \text{otherwise.} \end{cases}$$

Note that $\|w\|_2^2 = \frac{d}{2} \left(\frac{1}{\sqrt{\frac{2}{d}}} \right)^2 = \frac{d^2}{4}$. We also have that $h(p, w) = 1$ since $y\langle x, w \rangle = \sum_{i \in B_p} yx_i w_i = \frac{d}{2} 0 = 0$. To bound the sum of hinge losses contributed by other points $q \in P \setminus \{p\}$, note that $B_q \setminus B_p \neq \emptyset$. Then for every $q = (x', y') \neq p$,

$$y'\langle x', w \rangle = \sum_{i \in B_q \setminus B_p} y'x'_i w_i \geq \frac{1}{\sqrt{\frac{2}{d}}} \sqrt{\frac{2}{d}} = 1,$$

which implies that $h(q, w) = 0$. Thus,

$$\sum_{q \in P} h(q, w) = 1.$$

Putting it all together,

$$s(p) = \sup_{\substack{w' \in \mathbb{R}^{d+1} \\ F_\lambda(\mathcal{P}, w') \neq 0}} \frac{f_\lambda(p, w')}{F_\lambda(\mathcal{P}, w')} \geq \frac{\frac{d^2}{8n} + \lambda h(p, w)}{\frac{\|w\|_2^2}{2} + \lambda} = \frac{\frac{d^2}{8n} + \lambda}{\frac{d^2}{8} + \lambda}.$$

Since the above holds for every $p \in P$, summing the above inequality over every $p \in P$, yields that

$$\sum_{p \in P} s(p) \geq \frac{\frac{d^2}{8} + n\lambda}{\frac{d^2}{8} + \lambda} \in \Omega\left(\frac{d^2 + n\lambda}{d^2 + \lambda}\right).$$

□

Appendix A.2. Proof of Lemma 5

Lemma 5. *Let k be a positive integer, $\xi \in [0, F_\lambda(\mathcal{P}, w^*)]$, and let $\mathcal{P} = (P, u)$ be a weighted set. Then for every $i \in [k]$, $y \in \{+, -\}$ and $p \in P_y^{(i)}$,*

$$s(p) \leq \frac{u(p)}{U(P_y^{(i)})} + \lambda u(p) \frac{9}{2} \max \left\{ \frac{4}{9} \alpha_y^{(i)}, \sqrt{4 \left(\alpha_y^{(i)} \right)^2 + \frac{2 \|p_\Delta\|_2^2}{9 \widetilde{opt}_\xi}} - 2 \alpha_y^{(i)} \right\} = \gamma(p).$$

Proof. Let $P_y \subseteq P$ denote the set of points with the same label as p as in Line 4 of Algorithm 1. Consider an optimal clustering of the points in P_y into k clusters with centroids $\mathcal{C}^y = \{c_y^{(1)}, \dots, c_y^{(k)}\} \subseteq \mathbb{R}^{d+1}$ being their mean as in Line 5, and let $\alpha_y^{(i)}$ be as defined in Line 7 for every $i \in [k]$ and $y \in \{+, -\}$. In addition, let $\mathcal{P} \setminus \mathcal{P}_y^{(i)}$ denote the weighted set $(P \setminus P_y^{(i)}, u)$ for every $i \in [k]$ and $y \in \{+, -\}$.

Put $p = (x, y) \in P$ and let $i \in [k]$ be the index of the cluster which p belongs to, i.e., $p \in P_y^{(i)}$.

We first observe that for any scalars $a, b \in \mathbb{R}$, $\max\{a - b, 0\} \leq \max\{a, 0\} + \max\{-b, 0\}$. This implies that, by definition of the hinge loss, we have for every $q, \hat{q}, w \in \mathbb{R}^{d+1}$

$$h(q, w) \leq h(\hat{q}, w) + [\langle q - \hat{q}, w \rangle]_+,$$

where $[x]_+ = \max\{0, x\}$ as before. Hence, in the context of the definitions above

$$h(p, w) = h(p - c(p) + c(p), w) \quad (\text{A.1})$$

$$\leq h(c(p), w) + [\langle c(p) - yx, w \rangle]_+ \quad (\text{A.2})$$

$$= h(c(p), w) + [\langle p_\Delta, w \rangle]_+. \quad (\text{A.3})$$

Now let the total weight of the points in $P_y^{(i)}$ be denoted by $U(P_y^{(i)}) = \sum_{q \in P_y^{(i)}} u(q)$. Note that since $c_y^{(i)}$ is the centroid of $P_y^{(i)}$ (as described in Line 5 of Algorithm 1), we have $P_y^{(i)} = \frac{1}{U(P_y^{(i)})} \sum_{q=(x_q, y_q) \in P_y^{(i)}} u(q)y_q x_q$. Observing that the hinge loss is convex, we invoke Jensen's inequality to obtain

$$f_\lambda(c_y^{(i)}, w) \leq \frac{1}{U(P_y^{(i)})} \sum_{q \in P_y^{(i)}} u(q)f(q, w) = \frac{F_\lambda(\mathcal{P}, w) - F_\lambda(\mathcal{P} \setminus P_y^{(i)}, w)}{U(P_y^{(i)})}.$$

Applying the two inequalities established above to $s(p)/u(p)$ yields that

$$\frac{s(p)}{u(p)} = \sup_w \frac{f_\lambda(p, w)}{F_\lambda(\mathcal{P}, w)} \quad (\text{A.4})$$

$$\leq \sup_w \frac{f_\lambda(c_y^{(i)}, w) + \lambda[\langle w, p_\Delta \rangle]_+}{F_\lambda(\mathcal{P}, w)} \quad (\text{A.5})$$

$$\leq \sup_w \frac{\sum_{q \in P_y^{(i)}} u(q)f_\lambda(q, w)}{U(P_y^{(i)})F_\lambda(\mathcal{P}, w)} + \frac{\lambda[\langle w, p_\Delta \rangle]_+}{F_\lambda(\mathcal{P}, w)} \quad (\text{A.6})$$

$$= \sup_w \frac{F_\lambda(\mathcal{P}, w) - F_\lambda(\mathcal{P} \setminus P_y^{(i)}, w)}{U(P_y^{(i)})F_\lambda(\mathcal{P}, w)} + \frac{\lambda[\langle w, p_\Delta \rangle]_+}{F_\lambda(\mathcal{P}, w)} \quad (\text{A.7})$$

$$= \frac{1}{U(P_y^{(i)})} + \sup_w \frac{\lambda[\langle w, p_\Delta \rangle]_+ - F_\lambda(\mathcal{P} \setminus P_y^{(i)}, w)/U(P_y^{(i)})}{F_\lambda(\mathcal{P}, w)} \quad (\text{A.8})$$

By definition of $F_\lambda(\mathcal{P} \setminus P_y^{(i)}, w)$, we have

$$F_\lambda(\mathcal{P} \setminus P_y^{(i)}, w) \geq \frac{\|w_{1:d}\|_2^2 U(\mathcal{P} \setminus P_y^{(i)})}{2U(\mathcal{P})}.$$

Continuing from above and dividing both sides by λ yields

$$\begin{aligned}\frac{s(p)}{\lambda u(p)} &\leq \frac{1}{\lambda U(P_y^{(i)})} + \sup_w \frac{[\langle w, p_\Delta \rangle]_+ - \frac{\|w_{1:d}\|^2 U(P \setminus P_y^{(i)})}{2\lambda U(P)U(P_y^{(i)})}}{F_\lambda(\mathcal{P}, w)} \\ &\leq \frac{1}{\lambda u(\mathcal{C}_p)} + \sup_w \frac{[\langle w, p_\Delta \rangle]_+ - \alpha_y^{(i)} \|w_{1:d}\|_2^2}{F_\lambda(\mathcal{P}, w)},\end{aligned}$$

where

$$\alpha_y^{(i)} = \frac{U(P \setminus P_y^{(i)})}{2\lambda U(P)U(P_y^{(i)})}. \quad (\text{A.9})$$

Let

$$g(w) = \frac{[\langle w, p_\Delta \rangle]_+ - \alpha_y^{(i)} \|w_{1:d}\|_2^2}{F_\lambda(\mathcal{P}, w)}$$

be the expression on the right hand side of the sensitivity inequality above, and let $\hat{w} \in \operatorname{argmax}_w g(w)$. The rest of the proof will focus on bounding $g(\hat{w})$, since an upper bound on the sensitivity of a point as a whole would follow directly from an upper bound on $g(\hat{w})$.

Note that by definition of p_Δ and the embedding of 1 to the $(d+1)^{\text{th}}$ entry of the original d -dimensional point (with respect to p),

$$\langle \hat{w}, p_\Delta \rangle = \langle \hat{w}_{1:d}, (p_\Delta)_{1:d} \rangle,$$

where the equality holds since the $(d+1)^{\text{th}}$ entry of p_Δ is zero.

We know that $\langle \hat{w}, p_\Delta \rangle \geq \alpha_y^{(i)} \|\hat{w}_{1:d}\|_2^2 \geq 0$, since otherwise $g(\hat{w}) < 0$, which contradicts the fact that $\hat{w}$ is the maximizer of $g(w)$. This implies that for each entry $j \in [d]$ of the sub-gradient of $g(\cdot)$ evaluated at $\hat{w}$, denoted by $\nabla g(\hat{w})$, is given by

$$\nabla g(\hat{w})_j = \frac{\left((p_\Delta)_j - 2\alpha_y^{(i)} \hat{w}_j\right) F_\lambda(\mathcal{P}, \hat{w}) - \nabla F_\lambda(\mathcal{P}, \hat{w})_j \left(\langle \hat{w}, p_\Delta \rangle - \alpha_y^{(i)} \|\hat{w}_{1:d}\|_2^2\right)}{F_\lambda(\mathcal{P}, \hat{w})^2}, \quad (\text{A.10})$$

and that $\nabla g(\hat{w})_{d+1} = 0$ since the bias term does not appear in the numerator of $g(\cdot)$.

Letting $\gamma = \langle \hat{w}, p_\Delta \rangle - \alpha_y^{(i)} \|\hat{w}_{1:d}\|_2^2$ and setting each entry of the gradient $\nabla g(\hat{w})$ to 0, we solve for p_Δ to obtain

$$(p_\Delta)_{1:d} = \frac{\gamma \nabla F_\lambda(\mathcal{P}, \hat{w})_{1:d}}{F_\lambda(\mathcal{P}, \hat{w})} + 2\alpha_y^{(i)} \hat{w}_{1:d}.$$

This implies that

$$\langle \hat{w}, p_\Delta \rangle = \frac{\gamma \langle \hat{w}, \nabla F_\lambda(\mathcal{P}, \hat{w}) \rangle}{F_\lambda(\mathcal{P}, \hat{w})} + 2\alpha_y^{(i)} \|\hat{w}_{1:d}\|_2^2$$

Rearranging and using the definition of γ , we obtain

$$\gamma = \frac{\gamma \langle \hat{w}, \nabla F_\lambda(\mathcal{P}, \hat{w}) \rangle}{F_\lambda(\mathcal{P}, \hat{w})} + \alpha_y^{(i)} \|\hat{w}_{1:d}\|_2^2, \quad (\text{A.11})$$

where Lemma 5 holds by taking $\frac{9}{2}$ outside the max term.

By using the same equivalency for p_Δ from above, we also obtain that

$$\begin{aligned} \|p_\Delta\|_2^2 &= \langle p_\Delta, p_\Delta \rangle = \left\| \frac{\gamma \nabla F_\lambda(\mathcal{P}, \hat{w})_{1:d}}{F_\lambda(\mathcal{P}, \hat{w})} + 2\alpha_y^{(i)} \hat{w}_{1:d} \right\|^2 \\ &= \frac{\gamma^2}{F_\lambda(\mathcal{P}, \hat{w})^2} \|\nabla F_\lambda(\mathcal{P}, \hat{w})\|_2^2 + 4 \left(\alpha_y^{(i)} \right)^2 \|\hat{w}_{1:d}\|_2^2 + 4\alpha_y^{(i)} \frac{\gamma \langle \hat{w}, \nabla F_\lambda(\mathcal{P}, \hat{w}) \rangle}{F_\lambda(\mathcal{P}, \hat{w})}, \end{aligned}$$

but $\frac{\gamma \langle \hat{w}, \nabla F_\lambda(\mathcal{P}, \hat{w}) \rangle}{F_\lambda(\mathcal{P}, \hat{w})} = \gamma - \alpha_y^{(i)} \|\hat{w}_{1:d}\|_2^2$, and so continuing from above, we have

$$\begin{aligned} \|p_\Delta\|_2^2 &= \frac{\gamma^2}{F_\lambda(\mathcal{P}, \hat{w})^2} \|\nabla F_\lambda(\mathcal{P}, \hat{w})\|_2^2 + 4 \left(\alpha_y^{(i)} \right)^2 \|\hat{w}_{1:d}\|_2^2 + 4\alpha_y^{(i)} (\gamma - \alpha_y^{(i)} \|\hat{w}_{1:d}\|_2^2) \\ &= \frac{\gamma^2}{F_\lambda(\mathcal{P}, \hat{w})^2} \|\nabla F_\lambda(\mathcal{P}, \hat{w})_{1:d}\|_2^2 + 4\alpha_y^{(i)} \gamma \\ &= \gamma^2 \tilde{x} + 4\alpha_y^{(i)} \gamma, \end{aligned}$$

where $\tilde{x} = \frac{\|\nabla F_\lambda(\mathcal{P}, \hat{w})\|_2^2}{F_\lambda(\mathcal{P}, \hat{w})^2}$. Solving for γ from the above equation yields for $\tilde{x} > 0$

$$\gamma = \frac{\sqrt{4 \left(\alpha_y^{(i)} \right)^2 + \|p_\Delta\|^2 \tilde{x}} - 2\alpha_y^{(i)}}{\tilde{x}}. \quad (\text{A.12})$$

Now we subdivide the rest of the proof into two cases. The first is the trivial case in which the sensitivity of the point is sufficiently small enough to be negligible, and the second case is the involved case in which the point has a high influence on the SVM cost function and its contribution cannot be captured by the optimal solution w^* or something close to it.

Case $g(\hat{w}) \leq 3\alpha_y^{(i)}$ the bound on the sensitivity follows trivially from the analysis above.

Case $g(\hat{w}) > 3\alpha_y^{(i)}$ note that the assumption of this case implies that w^* cannot be the maximizer of $g(\cdot)$, i.e., $\hat{w} \neq w^*$. This follows by the convexity of the SVM loss function which implies that the norm of the gradient evaluated at w^* is 0. Thus by (A.11):

$$\gamma = \alpha_y^{(i)} \|w^*_{1:d}\|_2^2.$$

Since $F_\lambda(\mathcal{P}, w^*) \geq \|w^*_{1:d}\|_2^2/2$, we obtain

$$s(p) \leq \frac{\alpha_y^{(i)} \|w^*_{1:d}\|_2^2}{F_\lambda(\mathcal{P}, w^*)} \leq 2\alpha_y^{(i)}.$$

Hence, we know that for this case we have $\|\nabla F_\lambda(\mathcal{P}, \hat{w})\|_2 > 0$, $F_\lambda(\mathcal{P}, \hat{w}) > F_\lambda(\mathcal{P}, w^*) \geq 0$, and so we obtain $\tilde{x} > 0$.

This implies that we can use Eq. (A.12) to upper bound the numerator γ of the sensitivity. Note that γ from (A.12) is decreasing as a function of $\tilde{x}$, and so it suffices to obtain a lower bound on $\tilde{x}$. To do so, let's focus on Eq. (A.11) and let divide both sides of it by γ , to obtain that

$$1 = \frac{\langle \hat{w}, \nabla F_\lambda(\mathcal{P}, \hat{w}) \rangle}{F_\lambda(\mathcal{P}, \hat{w})} + \frac{\alpha_y^{(i)}}{\gamma} \|w_{1:d}\|_2^2.$$

By rearranging the above equality, we have that

$$\frac{\langle \hat{w}, \nabla F_\lambda(\mathcal{P}, \hat{w}) \rangle}{F_\lambda(\mathcal{P}, \hat{w})} = 1 - \frac{\alpha_y^{(i)} \|w_{1:d}\|_2^2}{\gamma}. \quad (\text{A.13})$$

Recall that since the last entry of p_Δ is 0 then it follows from Eq. (A.10) that $\nabla F_\lambda(\mathcal{P}, \hat{w})_{d+1}$ is also zero, which implies that

$$\begin{aligned} \langle \hat{w}, \nabla F_\lambda(\mathcal{P}, \hat{w}) \rangle &= \langle \hat{w}_{1:d}, \nabla F_\lambda(\mathcal{P}, \hat{w})_{1:d} \rangle \\ &\leq \|\hat{w}_{1:d}\|_2 \|\nabla F_\lambda(\mathcal{P}, \hat{w})_{1:d}\|_2 \\ &= \|\hat{w}_{1:d}\|_2 \|\nabla F_\lambda(\mathcal{P}, \hat{w})\|_2 \end{aligned} \quad (\text{A.14})$$

where the inequality is by Cauchy-Schwarz.

Combining Eq. (A.13) with Eq. (A.14) yields

$$\begin{aligned}
\frac{\|\hat{w}_{1:d}\|_2 \|\nabla F_\lambda(\mathcal{P}, \hat{w})\|_2}{F_\lambda(\mathcal{P}, \hat{w})} &\geq 1 - \frac{\alpha_y^{(i)} \|w_{1:d}\|_2^2}{\gamma} \\
&\geq 1 - \frac{\alpha_y^{(i)} \|\hat{w}_{1:d}\|_2^2}{3\alpha_y^{(i)} F_\lambda(\mathcal{P}, \hat{w})} \\
&\geq 1 - \frac{\alpha_y^{(i)} 2F_\lambda(\mathcal{P}, \hat{w})}{3\alpha_y^{(i)} F_\lambda(\mathcal{P}, \hat{w})} \\
&= \frac{1}{3},
\end{aligned}$$

where the second inequality holds by the assumption of the case, the third inequality follows from the fact that $\|\hat{w}_{1:d}\|_2^2 \leq 2F_\lambda(\mathcal{P}, \hat{w})$.

This implies that

$$\frac{\|\nabla F_\lambda(\mathcal{P}, \hat{w})\|_2}{F_\lambda(\mathcal{P}, \hat{w})} \geq \frac{1}{3\|w_{1:d}\|_2} \geq \frac{\sqrt{2}}{3\sqrt{F_\lambda(\mathcal{P}, \hat{w})}}.$$

Hence by definition of $\tilde{x}$, we have that

$$\tilde{x} \geq \frac{2}{9F_\lambda(\mathcal{P}, \hat{w})} \tag{A.15}$$

Plugging Eq. (A.15) into Eq. (A.12), we obtain that

$$\frac{\gamma}{F_\lambda(\mathcal{P}, \hat{w})} \leq \frac{9}{2} \left(\sqrt{4\left(\alpha_y^{(i)}\right)^2 + \frac{2\|p_\Delta\|_2^2}{9F_\lambda(\mathcal{P}, \hat{w})}} - 2\alpha_y^{(i)} \right).$$

Recall that

$$F_\lambda(\mathcal{P}, \hat{w}) \geq F_\lambda(\mathcal{P}, w^*) \geq F_\lambda(\mathcal{P}, \tilde{w}) - \xi,$$

which implies that

$$\frac{\gamma}{F_\lambda(\mathcal{P}, \hat{w})} \leq \frac{9}{2} \left(\sqrt{4\left(\alpha_y^{(i)}\right)^2 + \frac{2\|p_\Delta\|_2^2}{9\widetilde{opt}_\xi}} - 2\alpha_y^{(i)} \right), \tag{A.16}$$

where $\widetilde{opt}_\xi = F_\lambda(\mathcal{P}, \tilde{w}) - \xi$.

Combining both cases, yields that

$$s(p) \leq \frac{u(p)}{U(P_y^{(i)})} + u(p)\lambda \max \left\{ 2\alpha_y^{(i)}, \frac{9}{2} \left(\sqrt{4(\alpha_y^{(i)})^2 + \frac{2\|p_\Delta\|_2^2}{9\widetilde{opt}_\xi}} - 2\alpha_y^{(i)} \right) \right\}, \quad (\text{A.17})$$

where Lemma 5 holds by rearranging Eq. A.17 $\square$

Appendix A.3. Proof of Lemma 6

Lemma 6. *In the context of Lemma 5, the sum of sensitivities is bounded by*

$$\sum_{p \in P} s(p) \leq t = 4k + \sum_{i=1}^k \frac{3\lambda \text{Var}_+^{(i)}}{\sqrt{2\widetilde{opt}_\xi}} + \frac{3\lambda \text{Var}_-^{(i)}}{\sqrt{2\widetilde{opt}_\xi}},$$

where $\text{Var}_y^{(i)} = \sum_{p \in P_y^{(i)}} u(p) \|p_\Delta\|_2$ for all $i \in [k]$ and $y \in \{+, -\}$.

Proof. We first observe that that

$$\sum_{p \in P} s(p) = \sum_{i \in [k]} \left(\sum_{p \in P_+^{(i)}} s(p) + \sum_{p \in P_-^{(i)}} s(p) \right).$$

Thus we will focus on the summing the sensitivity of the all the points whose label is positive. We note that

$$\sum_{i \in [k]} \sum_{p \in P_+^{(i)}} \frac{u(p)}{U(P_+^{(i)})} = \sum_{i=1}^k 1 = k. \quad (\text{A.18})$$

In addition, we observe that $\max\{a, b\} \leq a + b$ for every $a, b \geq 0$, which implies that for every $i \in [k]$ and $p \in P_+^{(i)}$,

$$\begin{aligned} & \max \left\{ 2\alpha_y^{(i)}, \frac{9}{2} \left(\sqrt{4(\alpha_y^{(i)})^2 + \frac{2\|p_\Delta\|_2^2}{9\widetilde{opt}_\xi}} - 2\alpha_y^{(i)} \right) \right\} \\ & \leq 2\alpha_y^{(i)} + \frac{9}{2} \left(\sqrt{4(\alpha_y^{(i)})^2 + \frac{2\|p_\Delta\|_2^2}{9\widetilde{opt}_\xi}} - 2\alpha_y^{(i)} \right). \end{aligned} \quad (\text{A.19})$$

Since $\sqrt{a+b} \leq \sqrt{a} + \sqrt{b}$ for every $a, b \geq 0$, we have for every $i \in [k]$ and

$$p \in P_-^{(i)},$$

$$\begin{aligned}
& \sqrt{4 \left(\alpha_y^{(i)} \right)^2 + \frac{2 \|p_\Delta\|_2^2}{9 \widetilde{opt}_\xi}} - 2\alpha_y^{(i)} \\
& \leq 2\alpha_y^{(i)} + \frac{\sqrt{2} \|p_\Delta\|_2}{3\sqrt{\widetilde{opt}_\xi}} - 2\alpha_y^{(i)} \\
& = \frac{\sqrt{2} \|p_\Delta\|_2}{3\sqrt{\widetilde{opt}_\xi}}.
\end{aligned} \tag{A.20}$$

Hence by combining Eq.(A.18), Eq.(A.19), and Eq.(A.20), we yield that

$$\begin{aligned}
\sum_{i \in [k]} \sum_{p \in P_+^{(i)}} s(p) & \leq k + \sum_{i \in [k]} \sum_{p \in P_+^{(i)}} 2\lambda \alpha_+^{(i)} + \frac{9}{2} \lambda u(p) \frac{\sqrt{2} \|p_\Delta\|_2}{3\sqrt{\widetilde{opt}_\xi}} \\
& = k + \sum_{i \in [k]} \sum_{p \in P_+^{(i)}} 2\lambda \alpha_+^{(i)} + \sum_{i \in [k]} \lambda \frac{3\text{Var}_+^{(i)}}{\sqrt{2\widetilde{opt}_\xi}} \\
& \leq 2k + \sum_{i \in [k]} \lambda \frac{3\text{Var}_+^{(i)}}{\sqrt{2\widetilde{opt}_\xi}},
\end{aligned}$$

where the inequality follows from definition of $\alpha_y^{(i)}$ for every $i \in [k]$ and $y \in \{+, -\}$ as defined in Eq.(A.9).

Since all of the previous arguments hold similarly for P_- , we obtain that

$$\sum_{p \in P} s(p) \leq 4k + \sum_{i \in [k]} \frac{3\text{Var}_+^{(i)}}{\sqrt{2\widetilde{opt}_\xi}} + \frac{3\text{Var}_-^{(i)}}{\sqrt{2\widetilde{opt}_\xi}}.$$

□

Appendix A.4. Proof of Theorem 7

Theorem 7. For any $\varepsilon \in (0, 1/2)$, $\delta \in (0, 1)$, let m be an integer satisfying

$$m \in \Omega \left(\frac{t}{\varepsilon^2} (d \log t + \log(1/\delta)) \right),$$

where t is as in Lem. 6. Invoking CORESET with the inputs defined in this context yields a ε -coreset $\mathcal{S} = (S, v)$ with probability at least $1 - \delta$ in $\mathcal{O}(nd + T)$ time, where T represents the computational complexity of obtaining an ξ -approximated solution to SVM and applying k -means++ on P_+ and P_- .

Proof. By Lemma 5 and Theorem 5.5 of [3] we have that the coresset constructed by our algorithm is an ε -coreset with probability at least $1 - \delta$ if

$$m \geq c \left(\frac{t}{\varepsilon^2} (d \log t + \log(1/\delta)) \right),$$

where we used the fact that the VC dimension of a SVMs in the case of a linear kernel is bounded $\dim(\mathcal{F}) \leq d + 1 = \mathcal{O}(d)$ [29], and c is a sufficiently large constant which can be determined using similar techniques to that of [30]. Moreover, note that the computation time of our algorithm is dominated by going over the whole weighted set $\mathcal{P}$ which takes $\mathcal{O}(n)$ and attaining an ξ -approximation to the SVM problem at Line 1 followed by applying k -means clustering as shown in Algorithm 1 which takes $\mathcal{O}(T)$ time. This implies that the overall time is $\mathcal{O}(nd + T)$. $\square$

Corollary 7. *Let $\mathcal{P}$ be a weighted set, $\varepsilon \in (0, \frac{1}{2})$ and let $\mathcal{S}$ be an ε -coreset with respect to $\mathcal{P}$. Let $w_{\mathcal{S}}^* = \operatorname{argmin}_{w \in \mathbb{R}^{d+1}} F_{\lambda}(\mathcal{S}, w)$ and let $w_{\mathcal{P}}^*$ be defined similarly with respect to $\mathcal{P}$.*

$$F_{\lambda}(\mathcal{P}, w_{\mathcal{P}}^*) \leq F_{\lambda}(\mathcal{P}, w_{\mathcal{S}}^*) \leq (1 + 4\varepsilon) F_{\lambda}(\mathcal{P}, w_{\mathcal{P}}^*)$$

Proof. By Theorem 7, $\mathcal{S} = (S, v)$ is an ε -coreset for $(\mathcal{P}, u)$ with probability at least $1 - \delta$, which implies that

$$\begin{aligned} F_{\lambda}(\mathcal{P}, w_{\mathcal{S}}^*) &\leq \frac{F_{\lambda}(\mathcal{S}, w_{\mathcal{S}}^*)}{1 - \varepsilon} \leq \frac{F_{\lambda}(\mathcal{S}, w_{\mathcal{P}}^*)}{1 - \varepsilon} \\ &\leq \frac{1 + \varepsilon}{1 - \varepsilon} F_{\lambda}(\mathcal{P}, w_{\mathcal{P}}^*) \leq (1 + 4\varepsilon) F_{\lambda}(\mathcal{P}, w_{\mathcal{P}}^*), \end{aligned}$$

where the first and third inequalities follow from (S, v) being an ε -coreset (see Definition 2), the second inequality holds by definition of $w_{\mathcal{S}}^*$, and the last inequality follows from the assumption that $\varepsilon \in (0, \frac{1}{2})$. $\square$

Sufficient Conditions. Theorem 7 immediately implies that, for reasonable ε and δ , coresets of poly-logarithmic (in n) size can be obtained if $d = \mathcal{O}(\text{polylog}(n))$, which is usually the case in our target Big Data applications, and if

$$\sum_{i=1}^k \frac{3\lambda \text{Var}_+^{(i)}}{\sqrt{2\widetilde{\text{opt}}_{\xi}}} + \frac{3\lambda \text{Var}_-^{(i)}}{\sqrt{2\widetilde{\text{opt}}_{\xi}}} = \mathcal{O}(\text{polylog}(n)).$$

For example, a value of $\lambda \leq \frac{\log n}{n}$ for the regularization parameter λ satisfies the sufficient condition for all data sets with points normalized such that they are contained within the unit ball. Note that the total sensitivity, which dictates how many samples are necessary to obtain an ε -coreset with probability at least $1 - \delta$ and in a sense measures the difficulty of the problem, increases monotonically with the sum of distances of the points from their label-specific means.

Appendix B. Extension to Streaming Settings

As a corollary to our main method, Alg. 2 extends the capabilities of any SVM solver, exact or approximate, to the streaming setting, where data points arrive one-by-one. Alg. 2 is inspired by [4, 5] and constructs a binary tree, termed the *merge-and-reduce tree*, starting from the leaves which represent chunks of the data stream points. For each stream of l points, we construct an ε -coreset using Algorithm 1, and then we add each resulted tuple to B_1 , a bucket which is responsible for storing each of the parent nodes of the leaves (Lines 1-6).

Note that for every $i > 1$, the bucket B_i will hold every node which is a root of a subtree of height i . Then, for every two successive items in each bucket B_i , for every $i \geq 1$, a parent node is generated by computing a coreset on the union of the coresets, which is then, added to the bucket B_{i+1} (Lines 7-11). This process is done till all the buckets are emptied other than B_h , that will contain only one tuple (S, v) which is set to be the root of the merge-and-reduce tree.

In sum, we obtain a binary tree of height $h = \Theta(\log(n))$ for a stream of n data points. Thus, at Lines 4 and 10, we have used error parameter $\varepsilon' = \varepsilon/(2\log(n))$ and failure parameter $\delta' = \delta/(2\log(n))$ in order to obtain ε -coreset with probability at least $1 - \delta$.

Coreset construction of size poly-logarithmic in n . In case of the total sensitivity being sub-linear in n where n denotes the number of points in P , which is obtained by Lemma 5, we provide the following theorem which constructs a $(1 + \varepsilon)$ -coreset of size poly-logarithmic in n .

Algorithm 2: STREAMING-CORESET($P, u, \ell, \lambda, \xi, k$)

Input : An input stream P in $\mathbb{R}^{d+1} \times \{-1, 1\}$ of n points, a leaf size $\ell > 0$, a weight function $u : P \rightarrow \mathbb{R}_{\geq 0}$, a regularization parameter $\lambda \in [0, 1]$, a positive integer k , and an approximation factor $\xi > 0$.

Output : A weighted set (S, v)

```

1  $B_i \leftarrow \emptyset$  for every  $1 \leq i \leq \infty$ ;
2  $h \leftarrow 1$ ;
3 for each set  $Q$  of consecutive  $2\ell$  points from  $P$  do
4    $(T, v) \leftarrow \text{CORESET}(Q, u, \lambda, \xi, k, \ell)$ ;  $j \leftarrow 1$ ;
5    $B_j \leftarrow B_j \cup (T, v)$ ;
6   for each  $j \leq h$  do
7     while  $|B_j| \geq 2$  do
8        $(T_1, u_1), (T_2, u_2) \leftarrow$  top two items in  $B_j$ ;
9       Set  $\tilde{u} : T_1 \cup T_2 \rightarrow [0, \infty)$  such that for every  $p \in T_1 \cup T_2$ ,
          
$$\tilde{u}(p) = \begin{cases} u_1(p) & p \in T_1, \\ u_2(p) & \text{otherwise} \end{cases};$$

10       $(T, v) \leftarrow \text{CORESET}(T_1 \cup T_2, \tilde{u}, \lambda, \xi, k, \ell)$ ;
11       $B_{j+1} \leftarrow B_{j+1} \cup (T, v)$ ;
12       $h \leftarrow \max\{h, j + 1\}$ ;
13 Set  $(S, v)$  to be the only item in  $B_h$ 
14 return  $(S, v)$ 

```

Lemma 8. Let $\varepsilon \in [\frac{1}{\log n}, \frac{1}{2}]$, $\delta \in [\frac{1}{\log n}, 1)$, $\lambda \in (0, 1]$, a weighted set (P, u) , $\xi \in [0, F_\lambda(\mathcal{P}, w^*)]$ where $w^* \in \operatorname{argmin}_{w \in \mathbb{R}^{d+1}} F_\lambda(\mathcal{P}, w)$. Let t denote the total sensitivity from Lemma 5 and suppose that there exists $\beta \in (0.1, 0.8)$ such that $t \in \Theta(n^\beta)$. Let $\ell \geq \max \left\{ 2^{\frac{\beta}{1-\beta}}, c \left(\frac{t \log^2(n)}{\varepsilon^2} (d \log t + \log(\log n / \delta)) \right) \right\}$ and let (S, v) be the output of a call to STREAMING-CORESET($P, u, \ell, \lambda, \xi, .$). Then (S, v)

is an ε -coreset of size

$$|S| \in (\log n)^{\mathcal{O}(1)}.$$

Proof. First we note that using Theorem 7 on each node in the merge-and-reduce tree, would attain that the root of the tree, i.e., (S, v) attains that for every w

$$(1 - \varepsilon)^{\log n} F_\lambda(\mathcal{P}, w) \leq F_\lambda((S, v), w) \leq (1 + \varepsilon)^{\log n} F_\lambda(\mathcal{P}, w),$$

with probability at least $(1 - \delta)^{\log n}$.

We observe by the properties of the natural number e ,

$$(1 + \varepsilon)^{\log n} = \left(1 + \frac{\varepsilon \log n}{\log n}\right)^{\log n} \leq e^{\varepsilon \log n},$$

which when replacing ε with $\varepsilon' = \frac{\varepsilon}{2 \log n}$ in the above inequality as done at Lines 4 and 10 of Algorithm 2, we obtain that

$$(1 + \varepsilon')^{\log n} \leq e^{\frac{\varepsilon}{2}} \leq 1 + \varepsilon, \tag{B.1}$$

where the inequality holds since $\varepsilon \in [\frac{1}{\log n}, \frac{1}{2}]$.

As for the lower bound, observe that

$$(1 - \varepsilon)^{\log n} \geq 1 - \varepsilon \log n,$$

where the inequality holds since $\varepsilon \in [\frac{1}{\log n}, \frac{1}{2}]$.

Hence,

$$(1 - \varepsilon')^{\log n} \geq 1 - \varepsilon' \log n = 1 - \frac{\varepsilon}{2} \geq 1 - \varepsilon.$$

Similar arguments holds also for the failure probability δ . What is left for us to do is setting the leaf size which will attain us an ε -coreset of size poly-logarithmic in n (the number of points in P).

Let $\ell \in (0, \infty)$ be the size of a leaf in the merge-and-reduce tree. We observe that a coreset of size poly-logarithmic in n , can be achieved by solving the inequality

$$\frac{2\ell}{2} \geq (2\ell)^\beta,$$

which is invoked when ascending from any two leafs and their parent node at the merge-and-reduce tree.

Rearranging the inequality, we yield that

$$\ell^{1-\beta} \geq 2^\beta.$$

Since $\ell \in (0, \infty)$, any $\ell \geq {}^{1-\beta}\sqrt{2^\beta}$ would be sufficient for the inequality to hold. What is left for us to do, is to show that when ascending through the merge-and-reduce tree from the leaves towards the root, each parent node can't be more than half of the merge of it's children (recall that the merge-and-reduce tree is built in a binary tree fashion, as depicted at Algorithm 2).

Thus, we need to show that,

$$2^{\sum_{j=1}^i \beta^j} \cdot \ell^{\beta^i} \leq \frac{2^{\sum_{k=0}^{i-1} \beta^k} \cdot \ell^{\beta^{i-1}}}{2} = 2^{\sum_{k=1}^{i-1} \beta^k} \cdot \ell^{\beta^{i-1}},$$

holds, for any $i \in [\lceil \log n \rceil]$ where $\log n$ is the height of the tree. Note that the left most term is the parent node's size and the right most term represents half the size of both parent's children nodes.

In addition, for $i = 1$, the inequality above represents each node which is a parent of leaves. Thus, we observe that for every $i \geq 1$, the inequality represents ascending from node which is a root of a sub-tree of height $i - 1$ to it's parent in the merge-and-reduce tree.

By simplifying the inequality, we obtain the same inequality which only addressed the leaves. Hence, by using any $\ell \geq 2^{\frac{\beta}{1-\beta}}$ as a leaf size in the merge and reduce tree, we obtain an ε -coreset of size poly-logarithmic in n . $\square$

Appendix C. The logic behind applying k -means clustering

First put in mind the bound from Lemma 5 and note that it was achieved by using k -means++ clustering as depicted at Algorithm 1. Following our analysis from Sec. Appendix A.2, we observe that we can simply use any k -partitioning of the dataset, instead of applying k -means clustering. Moreover, we can also simply choose $k = 1$ which translates to simply taking the mean of the labeled

points. Despite all of above, we did choose to use a clustering algorithm as well as having larger values for k and such decisions were inspired by the following observations:

- (i) k -means clustering aims to optimize the sum of squared distances between the points and their respected center, which on some level, helps in lowering the distance between points and their respected centers. Such observation leads to having tighter bound for the sensitivity of each point, consequently leading to lower coreset sizes; See Thm. [7](#).
- (ii) Having larger k also helps lowering the distance between a point and its respected center.
- (iii) k -means clustering acts as a trade-off mechanism between
 - the *raw contribution* of each point, which is translated into the weight of the point divided by the sum of the weights with respect to the cluster that each point is assigned to,
 - and the *actual contribution* of each point which is translated to the distance between each point and its respected center.

In light of the above, we observe that as k goes larger, the sensitivity of each point gets closer and closer to being simply the *raw contribution*, which in case of unweighted data set, is simply applying uniform sampling. Thus, for each weighted (P, u) , we simply chose $k = \log n$ where n denotes the total number of points in a P .

This observation helps in understanding how the sensitivities that we are providing for outliers and misclassified points actually quantifies the importance of such points. specifically speaking, as k goes larger the outliers would mostly be assigned to the same cluster and since in general there aren't much of these points, we end up giving higher sensitivities for such points (than others) due to the fact that their *raw contribution* increases as the size of the cluster, they belong to, decreases. When k isn't large enough to separate these points from

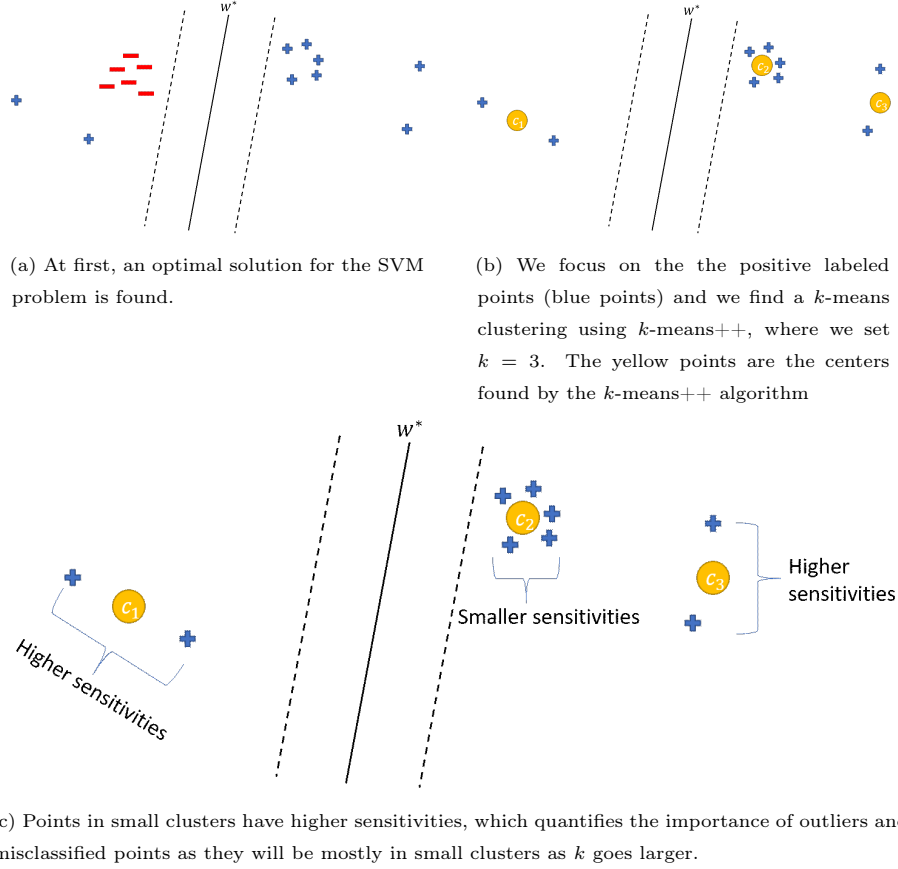


Figure C.4: Understanding the effect of k -means on the sensitivities of the points.

the rest of the data points, then the *actual contribution* kicks into play, which then the mean of the cluster is shifted towards the “middle” between the outliers and the rest of the points in that cluster, boosting the *actual contribution* of the rest of points inside the same cluster; See Fig. [C.4](#)

Appendix C.1. Towards finding the best k value

In the context of clustering, specifically speaking, k -means clustering problem, there is no definitive, provable way for determining the best k value, while considering the computation cost needed for applying the k -means algorithm (or alternatively k -means++). However, there are some tools which can point

to the best k value from a heuristic’s point of view. Such tools include the *Silhouette Method* and the *Elbow Method*. Fortunately enough, in our context, using the same methods can aid us in finding a “good” trade-off between the *raw contribution* and the *actual contribution*.

Although, such methods are proven to be useful in practice [26], in our experiments, we simply chosen a constant value for k depending on the number of points in the data set as elaborated in Section 6 and section Appendix D which is shown to be useful in practice at Figure 1 and Figure 3

Appendix D. Experimental Details

Our experiments were implemented in Python and performed on a 3.2GHz i7-6900K (8 cores total) machine with 64GB RAM. We considered the following datasets in our evaluations.

1. *HTRU* — 17,898 radio emissions, each with 9 features, of the Pulsar star.
2. *CreditCard* — 30,000 client entries each consisting of 24 features that include education, age, and gender among other factors.
3. *Pathological* — 1,000 points in two dimensional space describing two clusters distant from each other of different labels, as well as two points of different labels which are close to each other.³
4. *Skin* — 245,057 random samples of B,G,R from face images consisting of 4 dimensions.
5. *Cod(-rna)* — 488565 RNA records consisting each of 8 features.⁴
6. *W1* — 49,749 records of web pages consisting each of 300 features.

³We note that uniform sampling performs particularly poorly against this data set due to the presence of outliers.

⁴This data set was attained by merging the training, validation and testing sets.

Preprocessing step. Each data set has gone through a standardization process which aims to rescale the features so that they will have zero mean and unit standard deviation. As for the case where a data set is unweighted, we simply give each data point a weight of 1, i.e., $u : P \rightarrow 1$ where P denotes the data set, and the regularization parameter λ was set to be 1 throughout all of our experiments.

k-means clustering. In our experiments, we set $k = \log n$ where n is the number of points in the dataset (each datasets has different k value). As for the clustering itself, we have applied *k-means++* [31] on each of $\mathcal{P}_+$ and $\mathcal{P}_-$ as stated in our analysis; see Sec. 5.

Evaluation under streaming setting. Under streaming setting, the range for sample sizes is the same as for running under offline settings (See Figures 1 and 2). What differs is the quality of the solver itself, which we use to show the effectiveness of our coreset compared to uniform sampling, i.e., we have chosen to make the solver (SVC of Sklearn) more accurate by lowering its optimal tolerance.

Appendix E. Evaluations of Computational Cost for the Streaming Setting

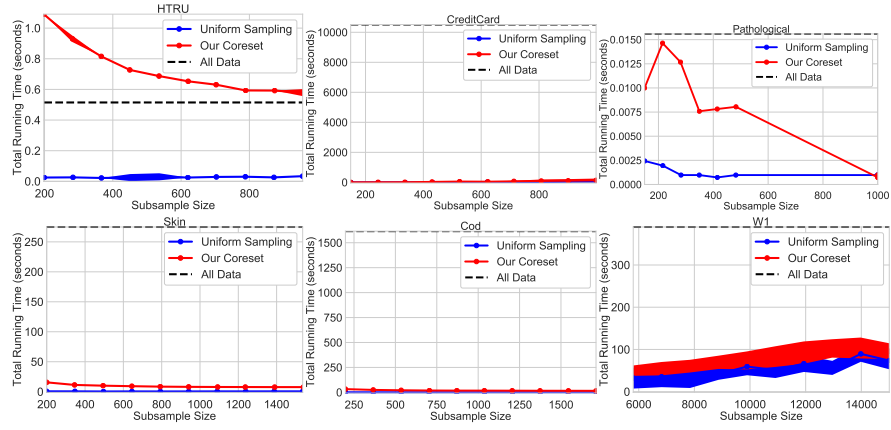


Figure E.5: The *total* computational cost of constructing a coreset using the merge-and-reduce tree and training the SVM model the coreset, plotted as a function of the size of the coreset.