

# Provable Imbalanced Point Clustering

David Denisov<sup>1</sup>, Dan Feldman<sup>1</sup>, Shlomi Dolev<sup>2</sup>, and Michael Segal<sup>2</sup>

<sup>1</sup> University of Haifa, Haifa, Israel.

E-mail: David Denisov: [daviddenisovphd@gmail.com](mailto:daviddenisovphd@gmail.com).

<sup>2</sup> Ben-Gurion University of the Negev, Beer-Sheva, Israel

**Abstract.** We suggest efficient and provable methods to compute an approximation for imbalanced point clustering, that is, fitting  $k$ -centers to a set of points in  $\mathbb{R}^d$ , for any  $d, k \geq 1$ . To this end, we utilize *coresets*, which, in the context of the paper, are essentially weighted sets of points in  $\mathbb{R}^d$  that approximate the fitting loss for every model in a given set, up to a multiplicative factor of  $1 \pm \varepsilon$ . We provide [Section 3 and Section E in the appendix] experiments that show the empirical contribution of our suggested methods for real images (novel and reference), synthetic data, and real-world data. We also propose choice clustering, which by combining clustering algorithms yields better performance than each one separately.

## 1 Introduction

Imbalanced clustering considers the problem of clustering data, with severe class distribution skews. This problem is of significant importance since this event can occur in practice, and if it does occur algorithms that do not support this, such as  $k$ -means, would give faulty results. An example of clustering with class imbalance and its effect on clustering via  $k$ -means (along with our proposed methods) is provided in Section 1.3.

**Previous work.** A common method to tackle the problem of imbalanced clustering is by under-sampling or over-sampling, where, informally, the points from the too-large or too-small clusters are sampled to obtain equal-sized clusters synthetically. To this end, the methods often require labels (or targets) for the points, which are not always given. Specifically, in Section E.1 we compare our methods to the implementation in [19] of works [23],[7],[18],[5], which all require labels (or targets). For a thorough review of such methods, see [29] and [16].

The problem of weighted clustering was previously considered, e.g. in [13], where each centroid point has its corresponding weight. Nonetheless, as in [13], the weights of all the centroid points fitted are often set before the fitting, while in our case the weight is a function of the cluster size and as such can not be preemptively set. Moreover, the coreset proposed in [13] is significantly larger than the coreset which we provide in Theorem 3; in [13] the coreset size has an exponential dependency on the number of clusters, while in our case the coreset size has a sub-quadratic dependency on the number of clusters.

**Novelty.** In contrast to the aforementioned prior works, we propose to solve the problem by considering loss functions that attempt to give each cluster equal importance. In particular, instead of attempting to minimize the total Euclidean distance to all the points (with square loss this is the goal of  $k$ -means), we attempt to minimize the sum (over the clusters) of the mean Euclidean distance to the points assigned to it. This is formalized in Section 1.2. As a result, our method does not require targets like most previous works ([23],[7],[18],[5], e.t.c.) and has essentially identical and even better performance as demonstrated in the additional tests at Section E of the appendix.

### 1.1 Paper structure

Our work has the following structure:

In Section 1.2 we state the objective functions that we aim to minimize and provide motivation for this task in Section 1.3. In Section 2 we state our main theoretical results, notably Lemma 1 and Theorem 1, whose proof is provided in the appendix.

In Section 3 we provide experimental results, mostly on images. Specifically: In Section 3.1 we demonstrate our results for image quantization of a “semi-synthetic” image (a photo of a rectangle drawn on paper). In Section 3.2 we provide a comparison to various clustering algorithms at Scikit-learn [25], inspired by a demonstration therein. In Section 3.3 we present *choice clustering* that entails computing clusters in various methods and choosing the best from the ones computed via some measure, such as the silhouette-score [28]. In Section 3.4 we demonstrate the effectiveness of *choice clustering* by comparing it to our suggested clustering and *k*-means, where those are also the clustering methods considered in the *choice clustering*, over real-world images (one provided by the authors and one a reference from the “USC-SIPI Image Database” <https://sipi.usc.edu/database/>).

In Section 4 we provide a conclusion to the paper, followed by Section 5 where we state future work and limitations.

In Section A we state our algorithms, divided as follows: Section A.1, where we state our proposed approximations to the loss function from Definition 1 in Algorithm 1, Section A.2, where we state our proposed coreset to the loss function from Definition 2 in Algorithm 2, and Section A.3, where we state the practical modifications in code done to our theoretically proven algorithms to improve their running time.

In Section B we prove Lemma 1, i.e., the desired properties of Definition 4.

In Section C we provide proof of the desired properties of Algorithm 1, split into Section C.1, where we state previous work on robust medians and prove efficient computation for our case, and Section C.2 where utilizing Section C.1 we prove desired properties of Algorithm 1.

In Section D we prove Theorem 1 that states the desired properties of Algorithm 2. To this end, we define and bound the VC-dimension of the problem in Section D.1, state previous works on the sensitivity of functions in Section D.2, and then prove Theorem 1 in Section D.

In Section E we provide additional tests of our methods both on synthetic and real-world, as follows. In Section E.1 we provide an extensive comparison of our methods to various clustering methods including (but not limited to) ones from the imbalanced-learn library [19] which require labels/targets, which are supplied. In Section E.2 we provide a comparison of our methods incorporated into hierarchical clustering and existing hierarchical clustering methods.

## 1.2 Objective functions

**Notations.** Throughout this paper we assume  $k, d \geq 1$  are integers, and denote by  $\mathbb{R}^d$  the union of  $d$ -dimensional real column vectors. A *weighted set* is a pair  $(P, w)$  where  $P$  is a finite set of points in  $\mathbb{R}^d$  and  $w : P \rightarrow [0, \infty)$  is a *weights function*. For simplicity, we denote  $\log(x) := \log_2(x)$ . In this work, we assume that all the input sets are finite and non-empty.

A method to define the problem of imbalanced clustering is to minimize the mean (over the clusters) of the cluster’s variance. That is formally stated as follows. Note that we define all the loss functions for general sets  $C \subset \mathbb{R}^d$ , but aim to minimize the loss over such sets of size at most  $k$ .

**Definition 1 (Loss function).** Let  $P \subset \mathbb{R}^d$  be a set of points. We define its fitting loss to a set  $C \subset \mathbb{R}^d$ , as

$$\ell(P, C) = \sum_{c \in C} \frac{1}{|P_c| + 1} \sum_{p \in P_c} \|p - c\|_2, \quad (1)$$

where  $\{P_c\}_{c \in C}$  is a partition of  $P$ , such that every  $p \in P$  is in  $P_i$  if  $c \in \arg \min_{c \in C} \|p - c\|_2$ . Ties broken arbitrarily. That is, for every  $c \in C$  we set  $P_c$  as the points in  $P$  that are closest to  $c$ . Our goal is to minimize  $\ell(P, C)$  over every set  $C$  of  $|C| = k$  centers.

To obtain a provable compression and more refined approximation, we introduce the following relaxation to the problem. The use of the following poly-logarithmic function enables a poly-logarithmic sized compression scheme (i.e., coreset, see Definition 5) for any given  $d$  and  $k$  (it is used in the proof of Theorem 7 specifically at E.q. [31–37]) while preserving that the impact of each point  $p$  on the loss is inversely (but not linearly) correlated to the size (in number of points) of the cluster that  $p$  belongs to.

**Definition 2 (Relaxed loss function).** Let  $P \subset \mathbb{R}^d$  be a set of points. We define the relaxed fitting loss of a set  $C \subset \mathbb{R}^d$ , as

$$\tilde{\ell}(P, C) = \sum_{c \in C} \frac{1}{\log^2(|P_c| + 1)} \sum_{p \in P_c} \|p - c\|_2, \quad (2)$$

where  $\{P_c\}_{c \in C}$  is a partition of  $P$ , such that every  $p \in P$  is in  $P_i$  if  $c \in \arg \min_{c \in C} \|p - c\|_2$ . Ties broken arbitrarily. That is, for every  $c \in C$  we set  $P_c$  as the points in  $P$  that are closest to  $c$ . Our goal is to minimize  $\tilde{\ell}(P, C)$  over every set  $C$  of  $|C| = k$  centers.

More generally, for a weighted set  $(C, w)$  we set the relaxed fitting loss of  $(C, w)$  as

$$\tilde{\ell}(P, (C, w)) = \sum_{c \in C} w(c) \frac{1}{\log^2(|P_c| + 1)} \sum_{p \in P_c} \|p - c\|_2, \quad (3)$$

where  $\{P_c\}_{c \in C}$  is a partition of  $P$  defined as above.

### 1.3 Motivation

In the following section, we demonstrate a motivation for the suggested clustering method, and its relaxation via comparison to  $k$ -means, specifically  $k$ -means++ [4], as implemented by Scikit-Learn [25]. This would be done by considering the problem of clustering points in  $\mathbb{R}^2$  where the clusters are very imbalanced, which would result in miss-classifying by  $k$ -means.

We utilized the following approximation, which is inspired by the centroid sets from [11], to the problem stated in Definition 1. For the properties of this method see Section 2.

**Definition 3.** Let  $(P, w)$  be a weighted set of size  $n \geq k$ . That is  $P \subset \mathbb{R}^d, |P| = n$  and  $w : P \rightarrow \mathbb{R}$ . Let  $\mathcal{P}_k$  be the union over all the subsets of size  $k$  from  $P$ . A set that minimized  $\tilde{\ell}((P, w), C)$  over every set  $C \in \mathcal{P}_k$  is called the optimal solution in  $\mathcal{P}_k$ . We denote by  $\text{opt}(P, w, k)$  this optimal solution in  $\mathcal{P}_k$ . Ties broken arbitrarily.

Simply replacing  $\tilde{\ell}$  by  $\ell$  in Definition 3 would not yield a provable approximation in general. Nonetheless, in this section, we approximate a minimizer to the loss function in Definition 1 via such substitution. In this section, for simplicity, we set  $k = 2$ .

We consider the following data sets in  $\mathbb{R}^2$ , where we use different values for  $n$ , which are the union of 2 uniform samples from discs generated as follows.

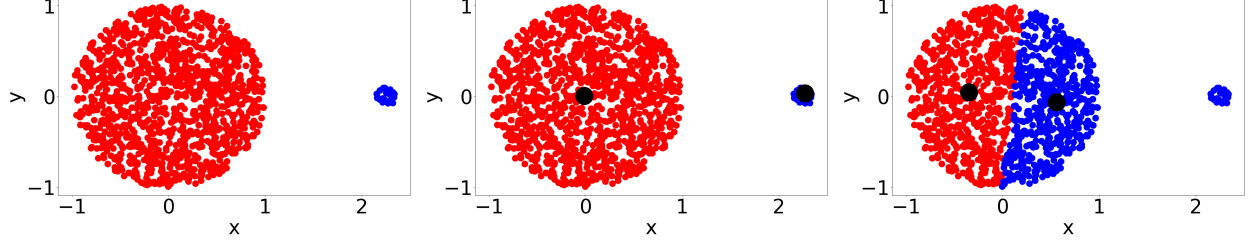
- (i). A sample of  $n$  “inliers” points, where each point is chosen uniformly inside the unit disc.
- (ii). A sample of 25 “outliers” points, where each point is chosen uniformly inside a disc of radius 0.1, centered at  $(2, 0)$ .

In Figure 1 we plot an example of the resulting the data generated, along with clustering provided by  $k$ -means and our solution to the objective function suggested in Definition 1.

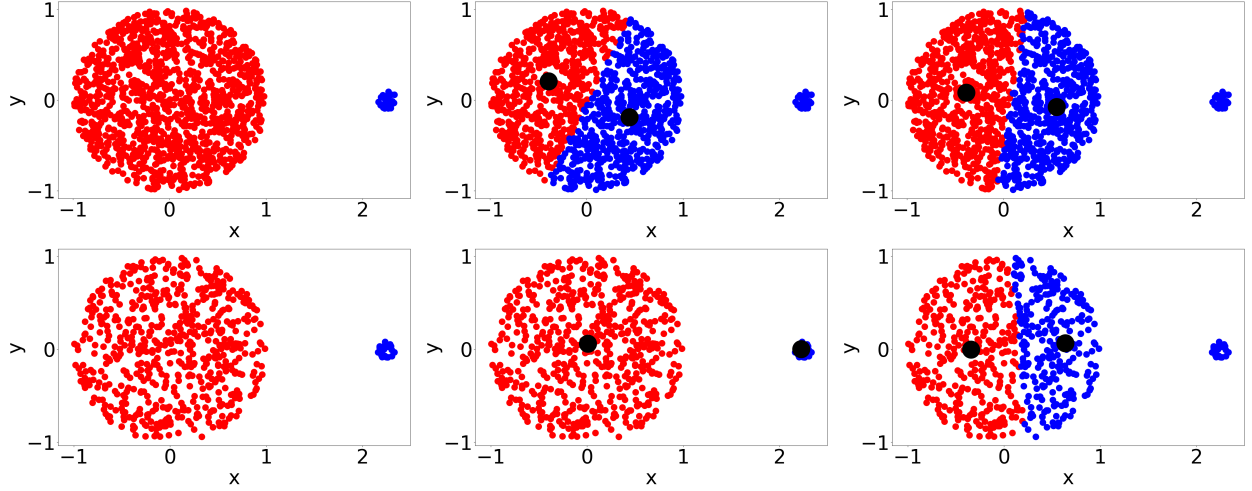
We had that  $k$ -means misclassified the original sample, and mixed part of the large cluster of “inliers” to the small one of “outliers”. On the other hand, our method has classified the clusters correctly. It should be emphasized that this is due to the different objectives, as while our method had a lower loss for our defined loss, it had a noticeably larger loss for the mean squared error that  $k$ -means++ attempts to minimize.

In the following test, we repeat the previous test, where we change our method to solve the problem in Definition 1 to a method to solve the problem in Definition 2. Results are provided in Figure 2.

For a large cluster size of 625 points, we have that our approximation to the problem suggested in Definition 2 does classify the data correctly, while  $k$ -means++ fails. However, when we increase the size of the larger cluster to 1250 (as in the previous example), our objective function as suggested in Definition 2 does not partition the data correctly. This demonstrates that, while the problem suggested in Definition 2 allows correct classification for imbalanced cases where  $k$ -means fails, it can still yield miss-classification for sufficiently large imbalances.



**Fig. 1.** Motivation for minimizing the loss function of Definition 1. **(left)** The left figure is the data generated for  $n = 1250$  “inliers” points and 25 “outliers”, the color of each point corresponds to the set from which it was generated (outlier or inlier). **(middle)** the black dots are the two optimal centers according to our approximation to the minimizer of the loss function in Definition Definition 1. The red points are closest to the first center, while the blue points are closest to the second center. **(middle)** the black dots are the two optimal centers according to our approximation to the minimizer of the loss function in Definition Definition 1. The red points are closest to the first center, while the blue points are closest to the second center. **((right))** the black dots are the two centers resulting by applying  $k$ -means++ [4] with  $k = 2$  for all the points. Again, the red points are closest to the first center, while the blue points are closest to the second center.



**Fig. 2.** Motivation for the problem suggested in Definition 2. The top row corresponds to  $n = 1250$  “inliers” points along 25 “outliers” points, and the bottom row to  $n = 625$  “inliers” points along 25 “outliers” points. The left columns are the data generated for the value of  $n$ , the color of each point corresponds to the set from which it was generated. In the right and middle columns, the black dots are the 2 centers computed, and the points of each cluster are colored depending on which center they are closest to (red or blue). The right figure demonstrates the output of  $k$ -means++ [4]. The middle figure demonstrates our approximations of the problem suggested in Definition 2.

## 2 Theoretical results

For presenting our theoretical results, we use the following notations and definitions.

**Definition 4 ( $\alpha$ -approximation).** Let  $\alpha \geq 1$  and  $P \subset \mathbb{R}^d$ . An  $\alpha$ -approximation  $C \subset \mathbb{R}^d$  of  $P$  is a set of size  $|C| = k$  such that

$$\tilde{\ell}(P, C) \leq \alpha \cdot \min_{C^* \subset \mathbb{R}^d, |C^*|=k} \tilde{\ell}(P, C^*). \quad (4)$$

A coresset for the fitting problem from Definition 2 is defined as follows.



**Definition 5 ( $\varepsilon$ -coreset).** Let  $P \subseteq \mathbb{R}^d$  and  $\varepsilon > 0$  be an error parameter. A pair  $(C, w)$ , where  $C \subset \mathbb{R}^d$  and  $w : C \rightarrow \mathbb{R}$ , is an  $\varepsilon$ -coreset of  $P$ , if, for every set  $C' \subset \mathbb{R}^d$  of size  $|C'| = k$ , we have

$$\left| \tilde{\ell}(P, C') - \tilde{\ell}((C, w), C') \right| \leq \varepsilon \cdot \tilde{\ell}(P, C'). \quad (5)$$

## 2.1 Main results

The following lemma states the main properties of Definition 4, which was inspired by the centroid sets method in [11]. For its proof see Lemma 2.

**Lemma 1.** Let  $(P, w)$  be a weighted set of size  $n \geq k$  where  $w : P \rightarrow [0, \infty)$ . That is  $P \subset \mathbb{R}^d$  and  $|P| = n$ . Suppose that  $\min_{C \subset \mathbb{R}^d, |C|=k} \tilde{\ell}((P, w), C)$  exists. Let  $Q := \text{APPROX}((P, w), k)$ ; see Definition 3. Then  $Q$  is a  $2 \log^2(1+n)$ -approximation for  $(P, w)$ ; see Definition 4. Moreover,  $Q$  can be computed in  $O(n^{k+1}dk)$  time.

The following theorem states the existence of an efficient coreset construction. Due to space limitations the formal statement, which states how to compute such a coreset, and the proof of the theorem are given at Theorem 3 in the appendix.

**Theorem 1.** There is an algorithm that gets  $P$ ,  $k$ ,  $\varepsilon \in (0, 1)$ ,  $\delta \in (0, 1/10]$ , and returns a weighted set  $(C, w)$  of size

$$|C| \in O\left(\frac{kd^3 \log(k) \log^4(n)}{\varepsilon^2} \left(\log(\log(k) \log(n)) + \log\left(\frac{1}{\delta}\right)\right)\right) \subseteq O\left(\frac{kd^3 \log^2 k \log^5 n + \log(1/\delta)}{\varepsilon^2}\right), \quad (6)$$

such that, with probability at least  $1-\delta$ ,  $C$  is an  $\varepsilon$ -coreset for  $P$ . Moreover,  $C$  can be computed in  $O(ndk \log(1/\delta))$  time.

## 3 Experimental results

Using Python 3.8, we implemented a coreset construction algorithm that satisfies Theorem 1 and the approximation from Lemma 1. Our changes from the theoretically proven algorithms are stated in Section A.3 after their pseudo-code. In all of the following tests, we use a PC with an Intel Core i5-12400F, NVIDIA GTX 1660 SUPER (GPU), and 32GB of RAM.

For a set  $P \subset \mathbb{R}^d$  of size  $|P| \geq k$  we denote  $\text{Approx-on-coreset}(P, k) := \text{Approx}((C, w), k)$ , where  $(C, w)$  is the weighted set stated in Theorem 1.

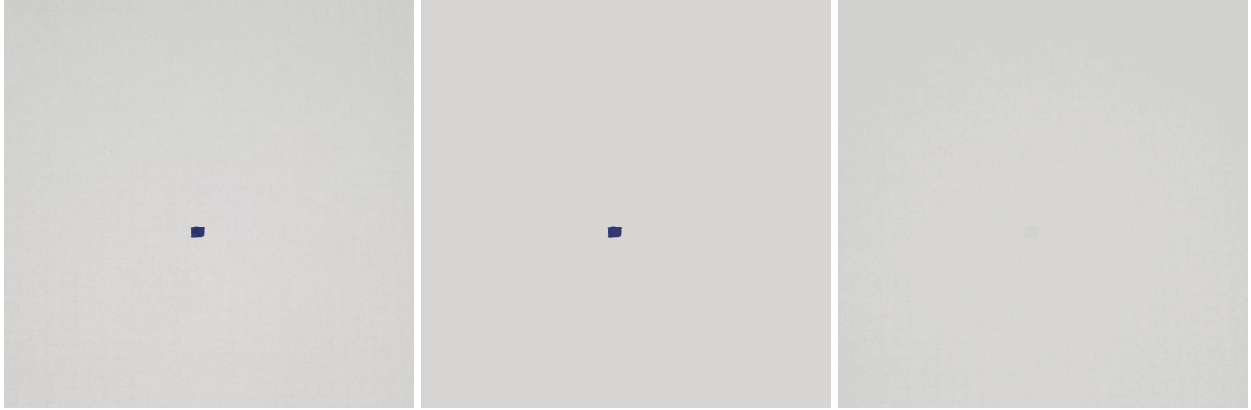
### 3.1 Real world motivation: image quantization

In the following example, we demonstrate that the previously stated class imbalance can occur in real-world tests. Specifically, we demonstrate this in the task of image quantization via clustering. It should be emphasized that our example can be quantized correctly via known existing methods. This would be done as in OpenCV's [6] K-Means Clustering in OpenCV tutorial, that is, paraphrasing the tutorial:

There are 3 features, say, R, G, and B. Reshape the  $n \times d \times 3$  image to a matrix of size  $M \times 3$  ( $M$  is the number of pixels in the image). Compute two clusters (also R, G, B), details are below. After the clustering, apply centroid values (it is also R, G, B) to all pixels, such that the resulting image will have a specified number of colors. Then reshape the result back to the shape of the original image, i.e., reshape to a  $n \times d \times 3$  matrix.

In the following example, we took an image of a white page with a small blue rectangle drawn on it. We consider two options to compute the two clusters stated in the quantization:

- (i). Our suggested clustering, which is provided by **Approx-on-coreset**.
- (ii). The  $k$ -means approximation clustering, which is applying  $k$ -means++ [4] as implemented by Scikit-Learn [25].



**Fig. 3.** Results for Section 3.1. The images are (left to right): the input image to cluster, the result of our quantization, and the result of the Scikit-Learn quantization. Note that the right image contains a small near-gray rectangular in place of the blue rectangle of the original image.

We refer to the image quantization corresponding to our clustering as our quantization, and similarly, to the image quantization corresponding to Scikit-Learn clustering as Scikit-Learn quantization. The results of the quantization are provided in Figure 3.

As can be seen, our quantization detected the rectangle as a distinct cluster from the page, while the Scikit-Learn quantization did not, and due to this it can barely be seen in the resulting quantization.

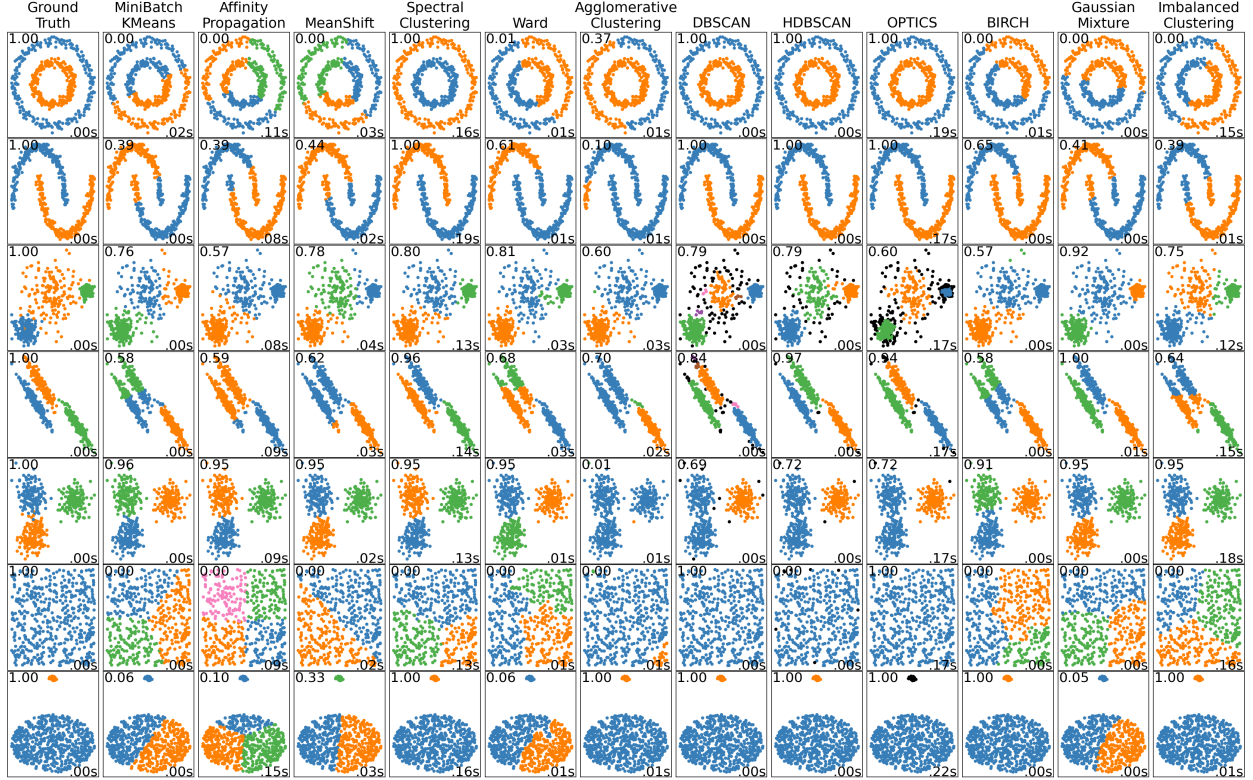
As stated in OpenCV’s [6] K-Means Clustering in OpenCV tutorial, a motivation for such quantization is to reduce the memory requirement of the images. The size of the produced image of our quantization is noticeably lower than the original image ( $\sim 20\text{kB}$  in “png” format v.s.  $\sim 500\text{kB}$  in “jpg” format and above  $3\text{MB}$  in “png” format), while no such noticeable decrease is observed for the Scikit-Learn quantization. This suggests that the competing clustering classified the “noise” of the image (for example, the inherent “noise” of the paper page, camera noise, etc.), which is mostly random and as such does not yield any compression.

### 3.2 Comparison to common clustering algorithms

Inspired by the comparison of various clustering algorithms at Scikit-learn [25], we compare our method to the ones presented therein. For a fair comparison, we utilized the open-source code for the comparison (note that the  $x$  and  $y$  values are presented differently than previously). Specifically, we added **Approx-on-coreset**, and the previously generated data at Section 1.3, for  $n := 625$  and with the same pseudo-random number generator as in the open source code. The results along with running times (bottom right) and V-measure [27] (top left), which attempts to measure accuracy (larger values are better), are presented in figure 4. Our method is in the rightmost column, and our additional data is in the bottom row.

Our method is essentially identical to the MiniBatch-KMeans in all the entries beside the bottom row, where it fails similarly to  $k$ -means in Section 1.3. That is for the top two rows the classification fails, as can be expected since the classes are not convexly separated. For the other rows as MiniBatch-KMeans, our method was successful, besides the central row (4 from top and bottom).

The only other methods that succeed for the bottom row are hierarchical clustering-based and spectral clustering. For hierarchical clustering and DBSCAN-related clustering (e.g., Ward to BIRCH), it can be seen that the methods fail, while ours does not, when the clusters are not well separated, such as rows 3,5,6 from the top. For spectral clustering, the method succeeds in all the tests, but note that the time complexity is very substantial and the method is prohibitively slow for large sets (for example  $100,000$  points) while  $k$ -means computes the output almost instantly for  $100,000$  points.



**Fig. 4.** Results for the comparison at Section 3.2. The rightmost column is our method and the bottom row is our motivation data from Section 1.3 for  $x := 625$ . The leftmost column is the ground truth clustering. The other rows were copied from the comparison at Scikit-learn [25], which this comparison is based on.

### 3.3 “Choice” clustering

Inspired by the non-decisive results of the compression at Section 3.2, where the best method (besides spectral clustering, which has large time complexity) for each dataset varies, we propose a novel method (at least to the best of our knowledge), of clustering via various algorithms and choose the best clustering among the results. In practice, we utilize the silhouette-score [28] to choose the best clustering but other methods can be used. To emphasize the effectiveness of the method we combine it with hierarchical clustering, as follows.

**Preliminaries on hierarchical clustering.** Hierarchical clustering consists of either merging or splitting clusters of the data, recursively, usually splitting into two clusters or merging pairs of clusters. See [24] an in-depth overview. Those algorithms are of significant use in practice, as demonstrated by their inclusion in Scikit-learn [25]. In this work, we mostly focus on the case of splitting clusters, or, as commonly referred to, divisive hierarchical clustering. The alternative method is merging clusters or as commonly referred to, agglomeration hierarchical clustering. For the divisive hierarchical clustering, we split each cluster into an equal number of clusters which are then split again up to a certain split depth, which, results in a balanced tree of a predefined depth. This method is rather similar to common algorithms for Decision trees, such as the CART algorithm [20]. There are many criteria for the split, notably  $k$ -means, with details in [17].

### 3.4 Examples

In the following example, we apply image quantization as explained in Section 3.1, with the only difference being that instead of the two clustering methods for the RGB values we consider the following three options

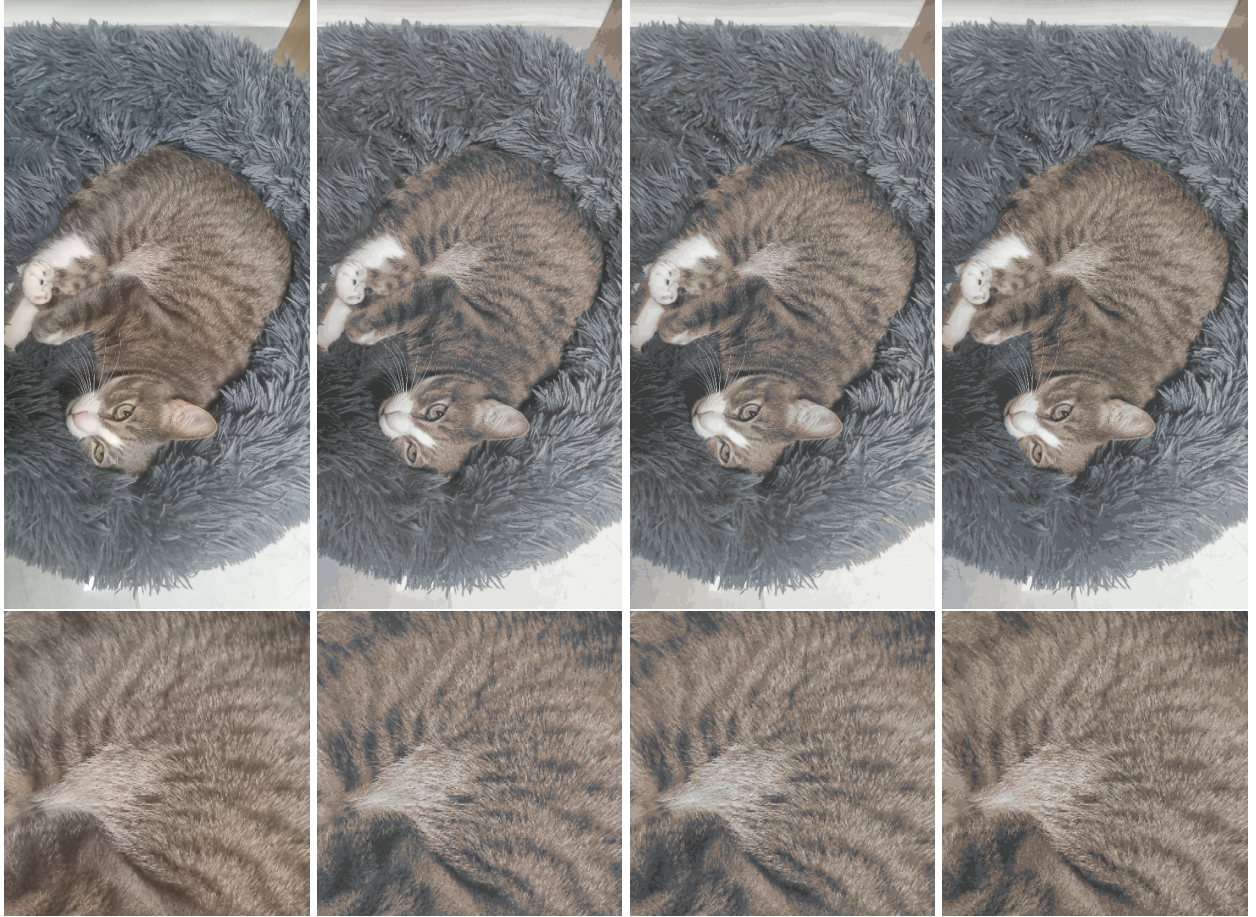


to compute the divisive clustering:

- (i). Our clustering, where each split in the divisive clustering is done by utilizing the **Approx-on-coreset** from Algorithm 2.
- (ii). The Scikit-Learn clustering, where each split in the divisive clustering is done by applying kmeans++ [4], as implemented by Scikit-Learn [25].
- (iii). Choice clustering, where each split in the divisive clustering is done by computing both the **Approx-on-coreset** clustering and the kmeans++ clustering, then choosing the clustering with the highest mean silhouette-score as implemented by Scikit-Learn with `sample_size` set to 1024. In case of an error in the computation of the silhouette-score, caused by its sample, we set `sample_size` to `None` that entitles no sample being done.

We refer to each quantization according to its clustering, that is, choice quantization is the quantization produced by choice clustering, etc.

In the following example, we have photographed a gray cat (Gray) lying in its blue cat bed, that is the image is novel. The image was resized to be  $1024 \times 512$  pixels. We consider a tree depth of 4, that is  $2^4 = 16$  levels. The results of the quantization are provided in Figure 5.



**Fig. 5.** Results for our original cat (Gray) image of Section 3.1. The rows correspond to **(top)**: The full images. **(bottom)**: a “zoom in” on a section of the cat’s fur for easier comparison. The images (left to right) are: the image we attempted to cluster (ground truth), the result of our quantization, the result of the Scikit-Learn quantization, and the result of the Choice quantization. Observe the cat’s fur at the ground truth image that has a blue hue in all the quantizations besides the Choice quantization.

In Figure 5, both our quantization and the Scikit-Learn quantization have resulted in a noticeable mis-coloring of the cat’s fur to have a bluish hue. In the Choice quantization, the cat’s fur (while still deviating from the ground truth) was noticeably closer to the original color.

The “USC-SIPI Image Database” contains (but is not limited to) a collection of various reference images commonly used for image processing. In the following example, we utilized the “Sailboat on lake” available at <https://sipi.usc.edu/database/preview/misc/4.2.06.png>. The image is of size  $512 \times 512$ , due to a rather noisy boundary we have removed the pixels across the image border. That is, the size of the considered image is  $510 \times 510$ . Then we quantized the image with a tree depth of 7, that is  $2^7 = 128$  levels. The results of the quantization are provided in Figure 6.



**Fig. 6.** Results for the boat image of Section 3.1. The images (left to right) are: the image we attempted to cluster (ground truth), the result of our quantization, the result of the Scikit-Learn quantization, and the result of the Choice quantization. Observe the red tree at the background of the ground truth image that is “missing” from all the quantizations besides the Choice quantization.

In Figure 6, both our quantization and the Scikit-Learn quantization have “missed” the red tree in the background and colored it similarly to the other trees. In the Choice quantization, the colors of the red tree in the background trees (while still deviating from the ground truth) were noticeably closer to the original colors. That is, when glancing over the images, the red tree is apparent only for Choice quantization among the quantization methods considered.

Those examples demonstrate the effectiveness of the proposed choice clustering, where the choice allowed better clustering than both the original clustering methods considered.

## 4 Conclusion

In this work, we presented a novel method for imbalanced clustering. This method supports unlabeled data, unlike the numerous previous methods that require labels and show no noticeable improvement over our method; see the additional tests in the appendix. Our method has competitive running time to [4], while not requiring equal-sized clusters for the classification as known previously and demonstrated in Figures [1,2]. We also proposed choice clustering and demonstrated significant improvement in image quantization over k-means, which is prevalent for this task.

We hope that our work will allow clustering for the numerous cases where despite there being no labels, it is also not the case that the clusters should be equally sized, e.g., as in Figures [1,2].

## 5 Future work and limitations

While the suggested approximation in Definition 4 performs rather well in practice we believe that there is a significant space for both practical and theoretical improvement. A consequence of this is that if there

is no significant class imbalance, that  $k$ -means would produce a better center choice, despite that the same clusters will (most likely) be chosen.

A limitation of our coresets construction is that it does not produce a weighted set as an output. This gives a substantial theoretical limitation to using the suggested coresets in an approximation. I.e., if the coresets returned a weighted set we could not only in practice but also in theory compute the provable approximation over the coresets. We hope to address this point in future revisions.

An additional limitation of our method is that it currently supports only Euclidean distances. We hope that our work will be generalized to absolute distances, robust M-estimators, and other loss functions that satisfy the triangle inequality, up to a constant factor, as in other coresets constructions [9].

## Acknowledgements

David Denisov was (partially) funded by the Israeli Science Foundation (Grant No. 465/22). Shlomi Dolev was (partially) funded by the Israeli Science Foundation (Grant No. 465/22) and by Rita Altura trust chair in Computer Science. Michael Segal was (partially) funded by the Israeli Science Foundation (Grant No. 465/22) and by the Army Research Office under Grant Number W911NF-22-1-0225. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

## References

1. Alexandrescu, A.: Fast deterministic selection. arXiv preprint arXiv:1606.00484 (2016)
2. Anagnostou, P., Tasoulis, S., Plagianakos, V.P., Tasoulis, D.: Hipart: Hierarchical divisive clustering toolbox. *Journal of Open Source Software* **8**(84), 5024 (2023). <https://doi.org/10.21105/joss.05024>, <https://doi.org/10.21105/joss.05024>
3. Anthony, M., Bartlett, P.L.: *Neural Network Learning: Theoretical Foundations*. Cambridge University Press (2009)
4. Arthur, D., Vassilvitskii, S.: K-means++: The advantages of careful seeding. In: *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*. p. 1027–1035. SODA '07, Society for Industrial and Applied Mathematics, USA (2007)
5. Batista, G.E., Bazzan, A.L., Monard, M.C., et al.: Balancing training data for automated annotation of keywords: a case study. *Wob* **3**, 10–8 (2003)
6. Bradski, G.: *The OpenCV Library*. Dr. Dobb's Journal of Software Tools (2000)
7. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research* **16**, 321–357 (2002)
8. Ding, Z.: *Diversified ensemble classifiers for highly imbalanced data learning and its application in bioinformatics*. Ph.D. thesis, Georgia State University (2011)
9. Feldman, D.: Core-sets: Updated survey. *Sampling Techniques for Supervised or Unsupervised Tasks* pp. 23–44 (2020)
10. Feldman, D., Kfir, Z., Wu, X.: Coresets for Gaussian mixture models of any shape. *CoRR* **abs/1906.04895** (2019), <http://arxiv.org/abs/1906.04895>
11. Feldman, D., Langberg, M.: A unified framework for approximating and clustering data. *Proceedings of the Annual ACM Symposium on Theory of Computing* (06 2011). <https://doi.org/10.1145/1993636.1993712>
12. Feldman, D., Schmidt, M., Sohler, C.: 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*. pp. 1434–1453. Society for Industrial and Applied Mathematics (2013)
13. Feldman, D., Schulman, L.J.: Data reduction for weighted and outlier-resistant clustering. In: *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*. pp. 1343–1354. Society for Industrial and Applied Mathematics (2012)
14. Har-Peled, S., Sharir, M.: Relative  $(p, \epsilon)$ -approximations in geometry. *CoRR* **abs/0909.0717** (2009), <http://arxiv.org/abs/0909.0717>



15. Jones, E., Oliphant, T., Peterson, P., et al.: SciPy: Open source scientific tools for Python (2001–), <http://www.scipy.org/>
16. Krawczyk, B.: Learning from imbalanced data: open challenges and future directions. *Progress in Artificial Intelligence* **5**(4), 221–232 (2016)
17. Lamrous, S., Taïleb, M.: Divisive hierarchical k-means. In: 2006 International Conference on Computational Intelligence for Modelling Control and Automation and International Conference on Intelligent Agents Web Technologies and International Commerce (CIMCA'06). pp. 18–18. IEEE (2006)
18. Last, F., Douzas, G., Bacao, F.: Oversampling for imbalanced learning based on k-means and smote (2017)
19. Lemaître, G., Nogueira, F., Aridas, C.K.: Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research* **18**(17), 1–5 (2017), <http://jmlr.org/papers/v18/16-365.html>
20. Lewis, R.J.: An introduction to classification and regression tree (cart) analysis. In: Annual meeting of the society for academic emergency medicine in San Francisco, California. vol. 14. Citeseer (2000)
21. Li, Y., Long, P.M., Srinivasan, A.: Improved bounds on the sample complexity of learning. *Journal of Computer and System Sciences* **62**(3), 516–527 (2001)
22. Lucic, M., Faulkner, M., Krause, A., Feldman, D.: Training gaussian mixture models at scale via coresets. *The Journal of Machine Learning Research* **18**(1), 5885–5909 (2017)
23. Mani, I., Zhang, I.: knn approach to unbalanced data distributions: a case study involving information extraction. In: Proceedings of workshop on learning from imbalanced datasets. vol. 126, pp. 1–7. ICML (2003)
24. Murtagh, F., Contreras, P.: Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* **2**(1), 86–97 (2012)
25. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* **12**, 2825–2830 (2011)
26. Raschka, S., Patterson, J., Nolet, C.: Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence. arXiv preprint arXiv:2002.04803 (2020)
27. Rosenberg, A., Hirschberg, J.: V-measure: A conditional entropy-based external cluster evaluation measure. In: Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL). pp. 410–420 (2007)
28. Rousseeuw, P.J.: Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics* **20**, 53–65 (1987). [https://doi.org/https://doi.org/10.1016/0377-0427\(87\)90125-7](https://doi.org/https://doi.org/10.1016/0377-0427(87)90125-7)
29. Santoso, B., Wijayanto, H., Notodiputro, K., Sartono, B.: Synthetic over sampling methods for handling class imbalanced problems : A review. *IOP Conference Series: Earth and Environmental Science* **58**, 012031 (03 2017). <https://doi.org/10.1088/1755-1315/58/1/012031>

## A Algorithms

For describing our algorithms, we use the following definitions. First, we extend the  $\alpha$ -approximation in Definition 4 to a more rough approximation, called  $(\alpha, \beta)$  or Bi-Criteria approximation [11].

**Definition 6 (( $\alpha, \beta$ )-approximation).** *Let  $\alpha \geq 0$ , and  $\beta \geq 1$  be an integer. Let  $P \subset \mathbb{R}^d$  be a finite set. An  $(\alpha, \beta)$ -approximation  $B := \{c_1, \dots, c_\beta\} \subset \mathbb{R}^d$  of  $P$  is a set of size  $k\beta$  such that*

$$\ell(P, B) \leq \alpha \cdot \min_{C \subset \mathbb{R}^d, |C|=k} \ell(P, C). \quad (7)$$

We define the following call to the approximation from [4].

*Claim (Theorem 3.1 [4]).* There is an algorithm that gets a finite set  $P \subseteq \mathbb{R}^d$  of size  $n \geq 1$ , and returns a set  $C \subseteq \mathbb{R}^d$  of  $|C| = k$  points (centers) such that, with probability at least  $1/2$ ,

$$\sum_{p \in P} \min_{c \in C} \|p - c\|_2 \in O(\log k) \cdot \min_{C' \subset \mathbb{R}^d, |C'|=k} \sum_{p \in P} \min_{c \in C'} \|p - c\|_2. \quad (8)$$

Moreover,  $C$  can be computed in  $O(dkn)$  time.

We denote this algorithm as KMEANS++, and it receives a finite set  $P \subseteq \mathbb{R}^d$  and the integer  $k$ .

### A.1 $(\alpha, \beta)$ -approximation

The input for the following algorithm is a set  $P \subset \mathbb{R}^d$  of size  $n$ , along with an integer  $k \geq 1$ , and an upper limit  $\delta \in (0, 1/10]$  on the allowed failure probability. Its output is a set  $B \subset \mathbb{R}^d$ , which is a  $O(k \log(n), k \log(n))$ -approximation of  $P$  with probability at least  $1 - \delta$ .

While the algorithm might initially look similar to the  $(\alpha, \beta)$ -approximation from [11], it does not entail solving the original problem on the sampled subset (we use a single and not  $k$  points as the sample). This modification enables a non-exponential dependency on  $k$ , unlike similar algorithms for  $k$ -means from [11]. It should be emphasized that there are approximation algorithms for  $k$ -means that are non-exponential dependency on  $k$ , notably [4], which is used in this work; see Algorithm 2.

The following theorem states the main properties of our algorithm. For its proof of correctness see Theorem 4.

**Theorem 2.** *Let  $P \subset \mathbb{R}^d$  be a set of size  $n \geq 2$ . Put  $\delta \in (0, 1/10]$ , and let  $B$  be the output of a call to  $\text{BI-CRITERIA}(P, k, \delta)$ ; see Algorithm 1. Then, with probability at least  $1 - \delta$ , we have that  $B$  is a  $O(k \log n, k \log n)$ -approximation to  $P$ . Moreover, the computation time of  $B$  is in  $O(ndk + dk^5 \log^3(n/\delta))$ .*

---

#### Algorithm 1: $\text{BI-CRITERIA}(P, k, \delta)$ ; see Theorem 2

---

**Input** : A finite set  $P \subset \mathbb{R}^d$  of size  $n \geq 2$ , an integer  $k \geq 1$ , and  $\delta \in (0, 1/10]$ .  
**Output**: A set  $B \subset \mathbb{R}^d$ , which, with probability at least  $1 - \delta$ , is an  $O(k \log n, k \log n)$ -approximation to  $P$ .

- 1  $P' := P$
- 2  $B := \emptyset$
- 3  $c :=$  a universal constant (in  $O(1)$ ) that can be derived from the poof of Theorem 2.
- 4  $\lambda := \lceil ck^2 \log(n/\delta) \rceil$ .
- 5 **while**  $|P'| \geq 2\lambda$  **do**
- 6     Pick a sample  $S \subseteq P$  of size  $\lambda$  from  $P'$ , where each element in  $S$  is sampled i.i.d. and uniformly at random from  $P'$ .
- 7     **for every**  $p \in S$  **do**
- 8         Set  $S_p$  as the subset of  $S$  of the  $\left\lfloor \frac{15|S|}{16k} \right\rfloor$  closest (euclidean distance) points to  $p$ , ties broken arbitrarily.
- 9          $\ell_p := \ell(S_p, \{p\})$
- 10     Let  $p \in \arg \min_{p \in S} \ell_p$ .
- 11     Set  $P'_p$  as the subset of  $P'$  of the  $\left\lfloor \frac{3|P'|}{4k} \right\rfloor$  closest (euclidean distance) points to  $p$ , ties broken arbitrarily.
- 12      $B = B \cup \{p\}$
- 13      $P' = P' \setminus P'_p$
- 14  $B = B \cup P'$
- 15 **return**  $B$

---

### A.2 Coreset

The following algorithm utilizes the sensitivity framework from [11] to compute an  $\varepsilon$ -coreset.

The following theorem states the main properties of the Algorithm, for its proof see Theorem 7.

**Theorem 3.** *Let  $P \subset \mathbb{R}^d$  be a finite set of size  $n \geq 2$ . Put  $\delta \in (0, 1/10]$ , and  $\varepsilon \in (0, 1)$ . Let  $(C, w)$  be the output of a call to  $\text{CORESET}(P, k, \delta, \varepsilon)$ ; see Algorithm 2. Then,*

$$|C| \in O\left(\frac{kd^3 \log(k) \log^4(n)}{\varepsilon^2} \left(\log(\log(k) \log(n)) + \log\left(\frac{1}{\delta}\right)\right)\right) \subseteq O\left(\frac{kd^3 \log^2 k \log^5 n + \log(1/\delta)}{\varepsilon^2}\right), \quad (9)$$



and, with probability at least  $1 - \delta$ , we have that  $(C, w)$  is a  $\varepsilon$ -coreset for  $P$ ; see Definition 5. Moreover,  $(C, w)$  can be computed in  $O(ndk \log(1/\delta))$  time.

---

**Algorithm 2:** CORESET( $P, k, \delta, \varepsilon$ ); see Theorem 3

---

**Input** : A set  $P \subset \mathbb{R}^d$  of size  $n \geq 2$ , integer  $k \geq 1$ ,  $\delta \in (0, 1/10]$ , and  $\varepsilon \in (0, 1)$ .  
**Output:** A pair  $(C, w)$ , where  $C \subset \mathbb{R}^d$  and  $w : C \rightarrow \mathbb{R}$ , which, with probability at least  $1 - \delta$  is an  $\varepsilon$ -coreset for  $P$ .

- 1 Set  $\lambda \in O\left(\frac{\log(k) \log^4(n)}{\varepsilon^2} \left(kd^3 \log(\log(k) \log(n)) + \log\left(\frac{1}{\delta}\right)\right)\right)$ ,  
where the exact value can be derived from the proof of Lemma 3.
- 2 **if**  $n \leq \lambda$  **then**
- 3     Set  $w : P \rightarrow \{1\}$ , i.e., the function that maps every  $p \in P$  to  $w(p) := 1$ .
- 4     **return**  $(P, w)$ .
- 5  $C := \emptyset$
- 6 **for every**  $i \in [\lceil \log(2/\delta) \rceil]$  **do**
- 7      $C_i := \text{KMEANS++}(P, k)$  // see Claim A,  $C_i$  is a set of  $k$  points in  $\mathbb{R}^d$ .
- 8      $C := C \cup \{C_i\}$  // note that  $C$  is a set of sets.
- 9 Let  $B \in \arg \min_{C \in \mathcal{C}} \sum_{p \in P} \min_{c \in C} \|c - p\|_2$ , ties broken arbitrarily //  $B$  is a set of  $k$  points in  $\mathbb{R}^d$ .
- 10 Let  $\mathcal{P} := \{P_c\}_{c \in B}$  be a partition of  $P$ , such that every  $p \in P$  is in  $P_c$  if  $c \in \arg \min_{c' \in B} \|c' - p\|_2$ . Ties broken arbitrarily; i.e., for every  $c \in B$  we set  $P_c$  are the points in  $P$  that are closest to  $c$ .
- 11 **for every**  $c \in B$  **do**
- 12     Set  $w(c) := |P_c|$ .
- 13     For every  $p \in P_c$ , set  $\ell_p := \frac{\|p - c\|_2}{\log^2(|P_c| + 1)}$
- 14 **if**  $\ell(P, B) = 0$  **then**
- 15     **return**  $(B, w)$
- 16 Set  $s(p) := \frac{\ell_p}{\sum_{p \in P} \ell_p}$  for every  $p \in P$ .
- 17 Pick a sample  $S$  of  $\lambda$  i.i.d. points from  $P$ , where each  $p \in P$  is sampled with probability  $s(p)$ .
- 18  $w(p) := 1/(\lambda \cdot s(p))$  for every  $p \in S$ .
- 19  $w(c) := w(c) - \frac{1}{\lambda \cdot s(p)}$  for every  $c \in B$  and  $p \in P_c$ .
- 20  $C := B \cup S$
- 21 **return**  $(C, w)$

---

### A.3 The gap between theory and practice

In our code, we apply minor changes to our theoretically provable algorithms, which have seemingly negligible effects on their output quality but aim to improve their running times. Those changes are:

(i). During Line 4 of Algorithm 1 we compute  $\lambda$  given the allowed probability of failure. Instead, we set  $\lambda := 64$  and fixing  $\delta$ .

(ii). Line 14 of Algorithm 1 sets  $B := B \cup P'$ . Instead, to reduce the size of  $B$  (which is the output of Algorithm 1), we set  $C := \text{APPROX}((P', w), k)$ , where  $w : P' \rightarrow \{1\}$  (each point is mapped to 1), and set  $B := B \cup C$ ; see Definition 3.

(iii). During Line 1 of Algorithm 2 we compute  $\lambda$  given the allowed probability of failure. Instead, we set  $\lambda := 128$  and fixing  $\delta$ .

(iv). In Lines[6–9] of Algorithm 2 we compute  $B \subset \mathbb{R}^d, |B| = k$  by rerunning  $\text{KMEANS++}(P, k)$   $O(\log(1/\delta))$  times. Again, we fix  $\delta$  so that the number iteration would be one, which amounts to setting  $B := \text{KMEANS++}(P, k)$ .

## B Constant factor approximation

In this section, we prove that we can compute an  $\alpha$ -approximation, as stated in Definition 3, for a set  $P \subset \mathbb{R}^d$  of size  $n \geq 2$ , and  $\alpha := 2^k \log^2(n+1)$ .

This is formally stated as follows, where the proof is inspired by [13].

**Lemma 2.** *Let  $(P, w)$  be a weighted set of size  $n \geq k$  where  $w : P \rightarrow [0, \infty)$ . That is,  $P \subset \mathbb{R}^d$  and  $|P| = n$ . Suppose that  $\min_{C \subset \mathbb{R}^d, |C|=k} \tilde{\ell}((P, w), C)$  exists. Let  $Q := \text{APPROX}((P, w), k)$ ; see Definition 3. Then  $Q$  is a  $2 \log^2(1+n)$ -approximation for  $(P, w)$ ; see Definition 4. Moreover, the computation time of  $Q$  is in  $O(n^{k+1}dk)$ .*

*Proof.* We will prove that there is a set  $C' \subset P$  of size  $k$ , which is a  $2^k \log^2(1+n)^2$ -approximation for  $(P, w)$ . Since we iterate in  $\text{APPROX}((P, w), k)$  over the sets  $C \subset P$  of size  $k$ , and take one of these subsets with the smallest loss, this proves that  $C$  is a  $2^k \log^2(1+n)$ -approximation for  $(P, w)$ .

Let  $C^* \in \arg \min_{C \subset \mathbb{R}^d, |C|=k} \tilde{\ell}((P, w), C)$ , which was assumed to exist. If  $C^* \subset P$ , then it immediately proves the existence of the stated  $C'$ . Hence, we assume this is not the case. That is,  $C^* \not\subset P$ . Let  $\mathcal{P} := \{P_c\}_{c \in C^*}$  be a partition of  $P$ , such that every  $p \in P$  is in  $P_c$  if  $c \in \arg \min_{c' \in C^*} \|c' - p\|_2$ . Ties broken arbitrarily; i.e., for every  $c \in C^*$  we set  $P_c$  as the points in  $P$  that are closest to  $c$ . For every  $c \in C^*$  let  $c_P$  be the closest (Euclidean distance) point in  $P$  to  $c$ . Ties broken arbitrarily.

Let  $c \in C^*$ . By the triangle inequality, for every  $p \in P$ , we have

$$\|c_P - p\|_2 \leq \|c_P - c\|_2 + \|c - p\|_2 \leq 2\|c - p\|_2. \quad (10)$$

Summing this over every  $p \in P_c$  yields

$$\sum_{p \in P_c} \|p - p_C\|_2 \leq 2 \sum_{p \in P_c} \|c - p\|_2. \quad (11)$$

Let  $C'' := \{c_P \mid c \in C^*\} \subset P$ . We have

$$\sum_{p \in P} w(p) \min_{c \in C''} \|p - c\|_2 \leq 2 \sum_{p \in P} w(p) \min_{c \in C^*} \|p - c\|_2. \quad (12)$$

Hence, by the choice of  $\tilde{\ell}$  in Definition 2, it follows that

$$\tilde{\ell}((P, w), C'') \leq 2 \log(1+n)^2 \tilde{\ell}((P, w), C^*), \quad (13)$$

which proves the existence of the sated  $C'$ .

**Running time.** Let  $C \subset \mathbb{R}^d$  be a set of size  $k$ . Observe that for each  $p \in P$  we can compute  $c \in \arg \min_C \|p - c\|_2$ , ties broken arbitrarily, in  $O(dk)$  time. Let  $f : P \rightarrow C$  that maps every  $p$  to its computed  $c \in \arg \min_C \|p - c\|_2$ . Observe that in the computation of  $\tilde{\ell}((P, w), C)$ , the mapping of  $f$  yields the desired partition of  $P$ . Hence, we can compute  $\tilde{\ell}((P, w), C)$  in  $O(ndk)$  time.

Observe that the number of sets of size  $k$  from  $P$  is in  $O(n^k)$ . Hence, we can compute the output of a call to  $\text{APPROX}((P, w), k)$  in  $O(n^{k+1}dk)$  time.

## C Efficient $(\alpha, \beta)$ -approximation

For our analysis of Algorithm 1 we require the following preliminaries on *robust medians*.

## C.1 Robust median

In the following section, we will define robust median and prove that we can compute it efficiently. Our method would be based on two parts, the initial method and its speed-up. The initial method entails taking the best center from a uniform sample of the points and is inspired by [13]. The speed-up entails computing the previous robust median on a uniform sample and is based on Section *From  $\varepsilon$ -approximation to robust medians* of [11].

**Initial method.** For the statement, we utilize the following definitions.

**Definition 7 ([13]).** Let  $P \subset \mathbb{R}^d$  be a non-empty finite set of points. For  $q \in \mathbb{R}^d$  and  $\gamma \in [0, 1]$ , we denote by  $\text{closest}(P, q, \gamma)$  the set that consists of the  $\lceil \gamma|P| \rceil$  points  $p \in P$  with the smallest values of  $\|q - p\|_2$ , ties broken arbitrarily.

**Definition 8 ([13]).** For  $\gamma \in [0, 1]$ ,  $\varepsilon \in (0, 1)$ , and  $\alpha > 0$ , the point  $q \in \mathbb{R}^d$  is a  $(\gamma, \varepsilon, \alpha)$ -median of the non-empty finite set  $Q \subset \mathbb{R}^d$  if

$$\sum_{p \in \text{closest}(Q, q, (1-\varepsilon)\gamma)} \|q - p\|_2 \leq \alpha \cdot \inf_{q' \in \mathbb{R}^d} \sum_{p \in \text{closest}(Q, q', \gamma)} \|q' - p\|_2. \quad (14)$$

Utilizing those definitions we can state the main result of our initial method to compute the robust median.

**Lemma 3.** Let  $P \subset \mathbb{R}^d$  be a finite and non-empty set. There is  $q \in P$ , which is an  $(15/16k, 1/16, 2)$ -median of  $P$ .

*Proof.* Let  $q^* \in \mathbb{R}^d$  be a  $(1/k, 0, 1)$ -median of  $P$ . Let  $q$  be the closest point in  $P$  to  $q^*$ . Ties are broken arbitrarily. By the triangle inequality, for every  $p \in P$ , we have

$$\|q - p\|_2 \leq \|q - q^*\|_2 + \|q^* - p\|_2 \leq 2\|q^* - p\|_2. \quad (15)$$

Let  $C^* := \text{closest}(P, q^*, 15/16k)$ . Summing this over every  $p \in \text{closest}(P, q, 15/16k)$  yields

$$\sum_{p \in \text{closest}(P, q, 15/16k)} \|q - p\|_2 \leq \sum_{p \in C^*} \|q - p\|_2 \leq 2 \sum_{p \in C^*} \|q^* - p\|_2. \quad (16)$$

Hence,  $q$  is a  $(15/16k, 0, 2)$ -median of  $P$ , which is also a  $(15/16k, 1/16, 2)$ -median of  $P$ .

Hence, as suggested by Lemma 3, our initial method to compute a robust median for a set is via an exhaustive search over all the points in the set.

**Speed up.** A consequence of Lemma 5.1 from [11] is the following result for our case.

**Lemma 4.** Let  $P \subset \mathbb{R}^d$  be a finite and non-empty set of points. Let  $\delta \in (0, 1/10]$ . Let  $S$  be a uniform random sample of i.i.d. points from  $P$ , of size  $\lambda := ck^2 \log(1/\delta)$ , for universal constant  $c$  that can be determined from the proof. With probability at least  $1 - \delta$ , any  $(15/16k, 1/16, 2)$ -median of  $S$  is a  $(1/k, 1/4, 2)$ -median of  $P$ .

## C.2 Analysis of Algorithm 1

In this section, we prove the Theorem 2.

**Theorem 4.** Let  $P \subset \mathbb{R}^d$  be a set of size  $n \geq 2$ . Put  $\delta \in (0, 1/10]$ , and let  $B$  be the output of a call to  $\text{BI-CRITERIA}(P, k, \delta)$ ; see Algorithm 1. Then, with probability at least  $1 - \delta$ , we have that  $B$  is a  $O(k \log n, k \log n)$ -approximation to  $P$ . Moreover, the computation time of  $B$  is in  $O(ndk + dk^5 \log^3(n/\delta))$ .

*Proof.* Let  $\lambda := bk^2 \log(n/\delta)$  where  $b$  is as defined in Lemma 4. Consider a single iteration of the “while” loop in the call to  $\text{BI-CRITERIA}(P, k, \delta)$ .

Substituting  $\delta := \delta/n$  in Lemma 4 yields that, with probability at least  $1 - \delta/n$ , any  $(15/16k, 1/16, 2)$ -median of  $S$  is a  $(1/k, 1/4, 2)$ -median of  $P'$ . Assume that this event indeed occurs.

By Lemma 3, there is  $p \in S$  which is a  $(15/16k, 1/16, 2)$ -median of  $S$ . Hence, by the definition of  $p \in S$  in the current “while” iteration it follows that  $p$  is a  $(15/16k, 1/16, 2)$ -median of  $S$ . Thus, we have that  $p$  is a  $(1/k, 1/4, 2)$ -median of  $P'$ .

Let  $P'_p$  be computed in this “while” iteration. We have that

$$\sum_{p' \in P'_p} \|p' - p\|_2 \leq 2 \min_{q^* \in \mathbb{R}^d} \sum_{p' \in \text{closest}(P, q^*, 1/k)} \|q^* - p\|_2. \quad (17)$$

Observe that due to the pigeonhole principle, for any possible  $k$  centers, there is a center with at least  $1/k$  of the points assigned to it. This yields

$$\frac{k}{|P| + 1} \cdot \min_{q^* \in \mathbb{R}^d} \sum_{p' \in \text{closest}(P, q^*, 1/k)} \leq \min_{C \subset \mathbb{R}^d, |C|=k} \ell(P, C). \quad (18)$$

Hence, we have that

$$\ell(P'_p, \{p\}) \leq 2 \min_{C \subset \mathbb{R}^d, |C|=k} \ell(P, C). \quad (19)$$

Observe that at each iteration of the “while” loop in Algorithm 1, the size of  $P'$  decreases by a multiplicative factor of  $1 - 1/(2k)$ . Hence, every  $2k$  iterations, the size of  $P'$  decreases by a multiplicative factor of  $(1 - 1/(2k))^{2k}$ . Let  $f : \mathbb{R} \rightarrow \mathbb{R}$  be the function that maps every  $x \in \mathbb{R}$  to  $f(x) := (1 - 1/x)^x$ . A known property of the function is that for every  $x \geq 1$ , we have that  $f(x) \leq 1/e$ . Hence, due to  $k$  being a positive integer we have that  $(1 - 1/(2k))^{2k} \leq 1/e$ . As such, for every  $2k$  iteration of the “while” loop, the size of  $P'$  decreases by a constant multiplicative factor. Hence, we have that there are  $O(k \log n)$  iterations in the “while” loop.

Therefore, since there are  $O(k \log n)$  iterations of the “while” loop, for  $B$  computed at the end of the “while” loop (Line 14 of Algorithm 1) we have

$$\ell(P \setminus P', B) \in O(k \log n) \min_{C \subset \mathbb{R}^d, |C|=k} \ell(P, C). \quad (20)$$

Thus since the algorithm returns  $B := B \cup P'$  it immediately follows that  $B$  is a  $O(k \log n, k \log n)$ -approximation to  $P$ .

We assumed, that at each iteration of the “while” loop, there is  $q \in S$  which is a  $(15/16k, 1/16, 2)$ -median of  $S$  and any such median is also a  $(1/k, 1/4, 2)$ -median of  $P'$ . This even occurs with a probability of at least  $1 - \delta/n$ . Hence, by the union bound and that  $\delta \leq 1/10$ , we have that all the even occurs with probability at least  $\left(1 - \frac{\delta}{n}\right)^n \leq 1 - \delta$ .

Thus, with probability at least  $1 - \delta$ , we have that  $B$  is a  $O(k \log n, k \log n)$ -approximation to  $P$ .

**Running time.** Consider a single iteration of the “while” loop in the call to  $\text{BI-CRITERIA}(P, k, \delta)$ .

It immediately follows that every “for” iteration can be done in  $O(\lambda d)$ ; note that the choice of  $S_p$  does not require sorting of the  $S$ , for implementation example see [1]. Hence, since there are  $\lambda$  iterations, it immediately follows that Lines 5–Lines 10 can be computed in  $O(\lambda^2 d)$  time. Hence, since  $\lambda \in O(k^2 \log(n/\delta))$ , Lines 5–Lines 10 can be computed in  $O(dk^4 \log^2(n/\delta))$  time.

Let  $n'$  be the size of  $P'$  in the “while” iteration considered. It can be seen that all the computations in Lines 11–Line 13 can be done in  $O(n'd)$  time; note that the choice of  $P'_p$  does not require sorting of the  $P$ , for implementation example see [1].

Hence, each “while” iteration in the call to  $\text{BI-CRITERIA}(P, k, \delta)$ , where  $n' := |P'|$ , can be computed in  $O(n'd + dk^4 \log^2(n/\delta))$  time.

Hence, since there are  $O(k \log n)$  iterations in the “while” loop, and in each iteration the size of  $P'$  decreases by a factor of  $\left(1 - \frac{1}{2k}\right)$ , i.e., we have a geometric sum, the total time of the “while” loop is in

$$\frac{O(nd)}{1 - \left(1 - \frac{1}{2k}\right)} + O(k \log n) \cdot dk^4 \log^2(n/\delta). \quad (21)$$

Hence, the total time of the “while” loop is in  $O(ndk + dk^5 \log^3(n/\delta))$ . Thus, by the construction of Algorithm 1, it follows that the computation time is dominated by the computation time of the “while” loop. That is, the call to  $\text{BI-CRITERIA}(P, k, \delta)$  can be computed in  $O(ndk + dk^5 \log^3(n/\delta))$  time.

## D Coreset for the relaxed problem

The coreset construction that we use in Algorithm 2 is a non-uniform sample from a distribution, which is known as sensitivity, that is based on the  $(\alpha, \beta)$ -approximation defined in Definition 4. To apply the generic coreset construction we need two ingredients:

1. A bound on the dimension induced by the query space (“complexity”) that corresponds to our problem as formally stated and bounded in subsection D.1. This bound determines the required size of the random sample picked in Algorithm 2.
2. A bound on the sensitivity as formally stated and bounded in the proof of Lemma 3. This bound on the sensitivity determines the required size of the random sample that is picked in Algorithm 2.

### D.1 Bound on the VC-Dimension

We first define the classic notion of VC-dimension, which is used in Theorem 8.14 in [3], and is usually related to the PAC-learning theory [21]. The following definition is from [22].

**Definition 9.** Let  $F \subset \{\mathbb{R}^d \rightarrow \{0, 1\}\}$  and let  $X \subset \mathbb{R}^d$ . Fix a set  $S = \{x_1, \dots, x_n\} \subset X$  and a function  $f \in F$ . We call  $S_f = \{x_i \in S \mid f(x_i) = 1\}$  the induced subset of  $S$  by  $f$ . A subset  $S = \{x_1, \dots, x_n\}$  of  $X$  is shattered by  $F$  if  $|\{S_f \mid f \in F\}| = 2^n$ . The VC-dimension of  $F$  is the size of the largest subset of  $X$  shattered by  $F$ .

**Theorem 5.** Let  $h$  be a function from  $\mathbb{R}^m \times \mathbb{R}^d$  to  $\{0, 1\}$ , and let

$$\mathcal{H} = \{h_\theta : \mathbb{R}^d \rightarrow \{0, 1\} \mid \theta \in \mathbb{R}^m\}. \quad (22)$$

Suppose that  $h$  can be computed by an algorithm that takes as input the pair  $\theta \in \mathbb{R}^m \times \mathbb{R}^d$  and returns  $h_\theta(x)$  after no more than  $t$  of the following operations:

- the arithmetic operations  $+$ ,  $-$ ,  $\times$ , and  $/$  on real numbers,
- jumps conditioned on  $>$ ,  $\leq$ ,  $<$ ,  $\geq$ ,  $=$ , and  $\neq$  comparisons of real numbers, and
- output 0, 1.

Then the VC-dimension of  $\mathcal{H}$  is  $O(m^2 + mt)$ .

For the sample mentioned at the start of Section D, we utilize the following generalization of the previous definition of VC-dimension. This is commonly referred to as VC-dimension, but to differentiate this definition from the previous, and to be in line with the notations in [10] we abbreviate it to *dimension*. This is the dimension induced by the query space which would be assigned in Theorem 6 to obtain the proof of desired properties of Algorithm 2.

**Definition 10 (range space [12]).** A range space is a pair  $(L, \text{ranges})$  where  $L$  is a set, called ground set and  $\text{ranges}$  is a family (set) of subsets of  $L$ .

**Definition 11 (dimension of range spaces [12]).** The dimension of a range space  $(L, \text{ranges})$  is the size  $|S|$  of the largest subset  $S \subseteq F$  such that

$$|\{S \cap \text{range} \mid \text{range} \in \text{ranges}\}| = 2^{|S|}. \quad (23)$$

**Definition 12 (range space of functions [12],[14],[11]).** Let  $F$  be a finite set of functions from a set  $\mathcal{Q}$  to  $[0, \infty)$ . For every  $Q \in \mathcal{Q}$  and  $r \geq 0$ , let  $\text{range}(F, Q, r) = \{f \in F \mid f(Q) \geq r\}$ . Let  $\text{ranges}(F) = \{\text{range}(F, Q, r) \mid Q \in \mathcal{Q}, r \geq 0\}$ . Let  $\mathcal{R}_{\mathcal{Q}, F} = (F, \text{ranges}(F))$  be the range space induced by  $\mathcal{Q}$  and  $F$ .

In the following lemma, which is inspired by Theorem 12 in [22], we bound the VC-dimension which would be assigned in Theorem 6 to obtain the proof of Algorithm 2.

**Lemma 5.** Let  $\{p_1, \dots, p_n\} := P \subset \mathbb{R}^d$ , and let  $\mathcal{Q}$  be the union over all the sets of size  $k$  in  $\mathbb{R}^d$ . For every  $i \in [n]$  let  $f_i : \mathcal{Q} \rightarrow [0, \infty)$  denote the function that maps every  $C \in \mathcal{Q}$  to  $f_i(C) = \frac{1}{|P_c''|} \min_{c \in C} \|p - c\|_2$ , where for every  $c \in C$  we set  $P_c''$  are the points in  $P$  that are closest to  $c$ . Let  $F = \{f_1, \dots, f_n\}$ . We have that the dimension of the range space  $\mathcal{R}_{\mathcal{Q}, F}$  is in  $O(kd^3)$ . The dimension of the range space  $\mathcal{R}_{\mathcal{Q}, F}$  that is induced by  $\mathcal{Q}$  and  $F$  is in  $O(kd^3)$ .

*Proof.* Let  $\mathcal{Q}'$  be the union of the weighted sets of size  $k$ . For every  $i \in [n]$  let  $f_i : \mathcal{Q} \rightarrow [0, \infty)$  denote the function that maps every  $C \in \mathcal{Q}'$  to  $f_i(C) := \min_{c \in C} w(c) \|p - c\|_2$ . It immediately follows that the dimension of the range space  $\mathcal{R}_{\mathcal{Q}, F'}$  is upper bound by the dimension of the range space  $\mathcal{R}_{\mathcal{Q}', F'}$ . Hence, from now on we prove that the dimension of the range space  $\mathcal{R}_{\mathcal{Q}', F'}$  is in  $O(kd^3)$ .

Let  $f' : \mathcal{Q}' \rightarrow \mathbb{R}^{d^2+d}$  be the function that maps every  $(C, w) \in \mathcal{Q}'$ ,  $\{c_i\}_{i=1}^d := C$  to  $f(C, w) := (c_1|c_2|\dots|c_d) \mid \{w(c)\}_{c \in C}$ . For every  $(q, r) = ((C, w), r) \in \mathcal{Q}' \times \mathbb{R}$ , let  $h_{(f(C, w)|r)} : \mathbb{R} \rightarrow \{0, 1\}$  that maps every  $p \in P$  to  $h_{(f(C, w)|r)}(p) = 1$  if and only if  $\min_{c \in C} w(c) \|p - c\|_2 \leq r$ , and every  $p \in \mathbb{R}^d \setminus P$  to  $h_{(f(C, w)|r)}(p) = 0$ . Let  $\mathcal{H} = \{h_\theta \mid \theta \in \mathbb{R}^{d^2+d+1}\}$ . For every  $(C, w) \in \mathcal{Q}$ , and  $p \in \mathbb{R}^d$  we can calculate  $\min_{c \in C} w(c)^2 \|p - c\|_2^2$  with  $O(kd)$  arithmetic operations on real numbers and jumps conditioned on comparisons of real numbers.

Therefore, for every  $p \in P$  and any  $\theta \in \mathbb{R}^{d^2+d+1}$  we can calculate  $h_\theta(i)$  with  $O(kd)$  arithmetic operations on real numbers and jumps conditioned on comparisons of real numbers. Hence, substituting  $d := n$ ,  $m := d^2+d+1$ ,  $h := h$ ,  $\mathcal{H} := \mathcal{H}$ , and  $t \in O(kd)$  in Theorem 5 yields that the VC-dimension of  $\mathcal{H}$  is in  $O(kd^3)$ . Hence, by the construction of  $\mathcal{H}$  and the definition of range spaces in Definition 12, we have that the dimension of the range space  $\mathcal{R}_{\mathcal{Q}', F'}$  that is induced by  $\mathcal{Q}'$  and  $F'$  is in  $O(kd^3)$ .

## D.2 Sensitivity of functions

For the self-containment of this work, we state previous work on the sensitivity of functions.

**Definition 13 (query space [10]).** Let  $P \subset \mathbb{R}^d$  be a finite nonempty set. Let  $f : P \times \mathcal{Q} \rightarrow [0, \infty)$  and  $\text{loss} : \mathbb{R}^{|P|} \rightarrow [0, \infty)$  be a function. The tuple  $(P, \mathcal{Q}, f, \text{loss})$  is called a query space. For every  $q \in \mathcal{Q}$  we define the overall fitting error of  $P$  to  $q$  by

$$f_{\text{loss}}(P, q) := \text{loss}(f(p, q)_{p \in P}) = \text{loss}(f(p_1, q), \dots, f(p_{|P|}, q)).$$

**Definition 14 ( $\varepsilon$ -coreset [10]).** Let  $(P, \mathcal{Q}, f, \text{loss})$  be a query space as in Definition 13. For an approximation error  $\varepsilon > 0$ , the pair  $S' = (S, u)$  is called an  $\varepsilon$ -coreset for the query space  $(P, \mathcal{Q}, f, \text{loss})$ , if  $S \subseteq P$ ,  $u : S \rightarrow [0, \infty)$ , and for every  $q \in \mathcal{Q}$  we have

$$(1 - \varepsilon)f_{\text{loss}}(P, q) \leq f_{\text{loss}}(S', q) \leq (1 + \varepsilon)f_{\text{loss}}(P, q).$$

**Definition 15 (sensitivity of functions).** Let  $P \subset \mathbb{R}^d$  be a finite and nonempty set, and let  $F \subset \{P \rightarrow [0, \infty]\}$  be a possibly infinite set of functions. The sensitivity of every point  $p \in P$  is

$$S_{(P,F)}^*(p) = \sup_{f \in F} \frac{f(p)}{\sum_{p \in P} f(p)}, \quad (24)$$

where  $\sup$  is over every  $f \in F$  such that the denominator is positive. The total sensitivity given a sensitivity is defined to be the sum over these sensitivities,  $S_F^*(P) = \sum_{p \in P} S_{(P,F)}^*(p)$ . The function  $S_{(P,F)} : P \rightarrow [0, \infty)$  is a sensitivity bound for  $S_{(P,F)}^*$ , if for every  $p \in P$  we have  $S_{(P,F)}(p) \geq S_{(P,F)}^*(p)$ . The total sensitivity bound is then defined to be  $S_{(P,F)}(P) = \sum_{p \in P} S_{(P,F)}(p)$ .

The following theorem proves that a coreset can be computed by sampling according to the sensitivity of functions. The size of the coreset depends on the total sensitivity and the complexity (dimension) of the query space, as well as the desired error  $\varepsilon$  and probability  $\delta$  of failure.

**Theorem 6 (coreset construction [10]).** Let

- $P = \{p_1, \dots, p_n\} \subset \mathbb{R}^d$  be a finite and non empty set, and  $f : P \times \mathcal{Q} \rightarrow [0, \infty)$ .
- $F = \{f_1, \dots, f_n\}$ , where  $f_i(q) = f(p_i, q)$  for every  $i \in [n]$  and  $q \in \mathbb{R}^d$
- $d'$  be the dimension of the range space that is induced by  $\mathbb{R}^d$  and  $F$ .
- $s^* : P \rightarrow [0, \infty)$  such that  $s^*(p)$  is the sensitivity of every  $p \in P$ , after substituting  $P = P$  and  $F = \{f' : P \rightarrow [0, \infty) \mid \forall p \in P, q \in \mathbb{R}^d : f'(p) := f(p, q)\}$  in Definition 15, and  $s : P \rightarrow [0, \infty)$  be the sensitivity bound of  $s^*$ .
- $t = \sum_{p \in P} s(p)$ .
- $\varepsilon, \delta \in (0, 1)$ .
- $c > 0$  is a universal constant that can be determined from the proof.
- $\lambda \geq c(t+1)(d' \log(t+1) + \log(1/\delta))/\varepsilon^2$ .
- $w : P \rightarrow \{1\}$ , i.e. a function such that for every  $p \in P$  we have  $w(p) = 1$ .
- $(S, u)$  be the output of a call to CORESET-FRAMEWORK( $P, w, s, \lambda$ ) (Algorithm 1 in [10]).

Then the following holds

- With probability at least  $1 - \delta$ ,  $(S, w)$  is an subset- $\varepsilon$ -coreset of size  $|S| \leq \lambda$  for the query space  $(F, \mathcal{Q}, f, \|\cdot\|_1)$ ; see Definition 14.

### D.3 Proof of Theorem 3

In the following section, we prove Theorem 3 that states the desired properties of Algorithm 2.

**Theorem 7.** Let  $P \subset \mathbb{R}^d$  be a set of size  $n \geq 2$ . Put  $\delta \in (0, 1/10]$ , and  $\varepsilon \in (0, 1)$ . Let  $(C, w)$  be the output of a call to CORESET( $P, k, \delta, \varepsilon$ ); see Algorithm 2. Then,  $|C|$  is in

$$|C| \in O\left(\frac{kd^3 \log(k) \log^4(n)}{\varepsilon^2} \left(\log(\log(k) \log(n)) + \log\left(\frac{1}{\delta}\right)\right)\right) \subseteq O\left(\frac{kd^3 \log^2 k \log^5 n + \log(1/\delta)}{\varepsilon^2}\right), \quad (25)$$

and with probability at least  $1 - \delta$ , we have that  $(C, w)$  is a  $\varepsilon$ -coreset for  $P$ . Moreover, the computation time of  $C$  is in  $O(dkn \log(1/\delta))$ .

*Proof.* Let  $B$  as computed in the call to CORESET( $P, k, \delta, \varepsilon$ ). Let  $k' \in O(\log(k))$  as stated in Claim A. By Claim A, for every  $C_i$  computed in the call to CORESET( $P, k, \delta, \varepsilon$ ), with probability at least  $1/2$ , we have

$$\sum_{p \in P} \min_{c \in C_i} \|p - c\|_2 \leq k' \min_{C' \subset \mathbb{R}^d, |C'|=k} \sum_{p \in P} \min_{c \in C'} \|p - c\|_2. \quad (26)$$

Hence, the event that

$$\sum_{p \in P} \min_{c \in B} \|p - c\|_2 > k' \min_{C' \subset \mathbb{R}^d, |C'|=k} \sum_{p \in P} \min_{c \in C'} \|p - c\|_2, \quad (27)$$

occurs with probability at most  $(1/2)^{\lceil \log(2/\delta) \rceil} \leq \delta/2$ . Hence, with a probability of at least  $1 - \delta/2$ , we have that

$$\sum_{p \in P} \min_{c \in B} \|p - c\|_2 \leq k' \min_{C' \subset \mathbb{R}^d, |C'|=k} \sum_{p \in P} \min_{c \in C'} \|p - c\|_2. \quad (28)$$

Suppose this even indeed occurs.

Observe, by the choice of  $\tilde{\ell}$  in Definition 2, that for every  $Q \subset \mathbb{R}^d, |Q| = k$  we have

$$\tilde{\ell}(Q, P) \leq \sum_{p \in P} \min_{c \in Q} \|p - c\|_2 \leq \log^2(n) \tilde{\ell}(Q, P). \quad (29)$$

Hence,  $B$  is an  $O(\log(k) \log^2(n))$ -approximation for  $P$ .

Let  $\alpha \in O(\log(k) \log^2(n))$  such that  $B$  is an  $\alpha$ -approximation to  $P$ . Let  $s : P \rightarrow [0, \infty)$ , and  $\mathcal{P} := \{P_c\}_{c \in B}$  as computed in the call to  $\text{CORESET}(P, k, \delta, \varepsilon)$ . Let  $w' : B \rightarrow [0, \infty)$  that maps every  $c \in B$  to  $|P_c|$ .

Suppose  $\ell(B, P) = 0$ . That is for every  $c \in B$  and  $p \in P_c$  we have  $p = c$ . By the construction of Algorithm 2, we have that  $C = B$ , and for every  $c \in C$  we have  $w(c) = |P_c|$ . Hence, it immediately holds that  $(C, w)$  is an  $\varepsilon$ -coreset for  $P$ . Therefore, from now on we assume this is not the case, i.e.,  $\ell(B, P) > 0$ .

Let  $\mathcal{Q}$  be the union over all the sets of size  $k$  in  $\mathbb{R}^d$ . Let  $Q \in \mathcal{Q}$ . We have

$$\left| \tilde{\ell}(P, Q) - \tilde{\ell}((C, w), Q) \right| = \tilde{\ell}(P, Q) \cdot \left| \frac{\tilde{\ell}(P, Q) - \tilde{\ell}((B, w'), Q)}{\tilde{\ell}(P, Q)} - \frac{\tilde{\ell}((C, w), Q) - \tilde{\ell}((B, w'), Q)}{\tilde{\ell}(P, Q)} \right|, \quad (30)$$

which follows from reorganizing the expression and taking  $\ell(P, Q)$  out of the sum.

Let  $\mathcal{P}' := \{P'_c\}_{c \in Q}$  as a partition of  $P'$  such that every  $p \in P$  is in  $P'_c$  if  $c \in \arg \min_{c' \in Q} \|c' - p\|_2$ . Ties broken arbitrarily; i.e., for every  $c \in Q$  we set  $P'_c$  are the points in  $P'$  that are closest to  $c$ . Consider  $q \in Q$ , where  $P'_q$  is non-empty.

Let  $s^* : P'_q \rightarrow [0, \infty)$  such that for every  $c \in B$  and  $p \in P_c \cap P'_q$  we have

$$s^*(p) = \frac{\left| \|q - p\|_2 - \|q - c\|_2 \right|}{\log^2(|P'_q| + 1) \cdot \tilde{\ell}(P, Q)}, \quad (31)$$

which, due to the definition of  $w'$ , is an upper bound on the contribution of every point in  $P'_q$  to the sum in Equation (30).

Let  $c \in B$  and  $p \in P_c \cap P'_q$ , so that

$$s^*(p) \leq \frac{\left| \|q - p\|_2 - \|q - c\|_2 \right|}{\log^2(|P'_q| + 1) \cdot \tilde{\ell}(P, Q)} \quad (32)$$

$$\leq \frac{\|p - c\|_2}{\log^2(|P'_q| + 1) \cdot \tilde{\ell}(P, Q)} \quad (33)$$

$$\leq \alpha \cdot \frac{\log^2(|P'_q| + 1)}{\log^2(|P_c| + 1)} \cdot s(p) \quad (34)$$

$$\leq \alpha \cdot \log^2(n + 1) \cdot s(p), \quad (35)$$

$$\leq 4\alpha \cdot \log^2(n) \cdot s(p), \quad (36)$$

where Equation 32 is by the definition of  $s^*$  from Equation (31), Equation (33) is by the reverse triangle inequality, Equation (34) is by the choice of  $s$  and the definition of  $B$  as an  $\alpha$ -approximation of  $P$ , Equation (35) is by reorganizing, and Equation (36) is by assigning that  $n \geq 2$ .



Let  $\tilde{s} : P \rightarrow [0, \infty)$  such that for every  $q \in Q$  and  $p \in P_q$  we have  $\tilde{s}(p) = 4\alpha \cdot \log^2 n \cdot s(p)$ . By Equations [32–34] we have that  $\tilde{s}$  is a sensitivity bound for  $s^*$ , for every  $q \in Q$ .

Let  $\{p_1, \dots, p_n\} := P$ , and for every  $i \in [n]$  let  $f_i : \mathcal{Q} \rightarrow [0, \infty)$  denote the function that maps every  $C' \in \mathcal{Q}$  to  $f_i(C') = \frac{1}{|P_c''|} \min_{c \in C'} \|p - c\|_2$ , where for every  $c \in C'$  we set  $P_c''$  are the points in  $P$  that are closest to  $c$ . Let  $F = \{f_1, \dots, f_n\}$ , and  $d' \in O(kd^3)$  be the dimension of the range space  $\mathcal{R}_{\mathcal{Q}, F}$  from Lemma 5 when assigning  $P$ .

Substituting the query space  $(P, \mathcal{Q}, F, \|\cdot\|_1)$ ,  $d' \in O(kd^3)$  the VC-dimension induced by  $\mathcal{Q}$  and  $F$  from Lemma 5, the sensitivity bound  $\tilde{s}$ , and the total sensitivity  $t = 4\alpha \log^2(n) \sum_{p \in P} s(p) = 4\alpha \log^2(n) \in O(\log(k) \cdot \log^4(n))$  in Theorem 6, combined with the construction of Algorithm 2, yields that there is  $\lambda$  as defined in the lemma such that with probability at least  $1 - \delta/2$ , for every  $Q \in \mathcal{Q}$

$$\left| \frac{\tilde{\ell}(P, Q) - \tilde{\ell}((B, w'), Q)}{\tilde{\ell}(P, Q)} - \frac{\tilde{\ell}((C, w), Q) - \tilde{\ell}((B, w'), Q)}{\tilde{\ell}(P, Q)} \right| \leq \varepsilon. \quad (37)$$

Combining Equation 37 and Equation 30 yields that with probability at least  $1 - \delta/2$  we have

$$\left| \tilde{\ell}(P, Q) - \tilde{\ell}((C, w), Q) \right| \leq \varepsilon \tilde{\ell}(P, Q). \quad (38)$$

That is,  $(C, w)$  is an  $\varepsilon$ -coreset.

Hence, by the union bound, we have with probability at least  $(1 - \delta/2)^2 \geq 1 - \delta$  we have that  $(C, w)$  is an  $\varepsilon$ -coreset.

**Size of  $C$ .** By its construction in the call to  $\text{CORESET}(P, k, \delta, \varepsilon)$ , we have that  $|B| = k$ . Recall the choice of  $\lambda$  in Line 1 of Algorithm 2, that is

$$\lambda \in O\left(\frac{\log(k) \log^4(n)}{\varepsilon^2} \left(kd^3 \log(kn) + \log\left(\frac{1}{\delta}\right)\right)\right). \quad (39)$$

The size of the sample  $S$  taken in Line 1 of Algorithm 2 is  $\lambda$ .

Hence, since  $C := S \cup B$  we have that the claim on the size of  $C$  holds.

**Running time.** By Claim A, we have that each call to  $\text{KMEANS}++(P, k)$  can be computed in  $O(ndk)$  time. Hence, since there are  $\lceil \log(2/\delta) \rceil$  calls to  $\text{KMEANS}++(P, k)$ , we have that  $B$  can be computed in  $O(ndk \log(1/\delta))$  time. All the other computations in the call to  $\text{CORESET}(P, k, \delta, \varepsilon)$  can be computed in  $O(ndk)$  time. Hence, the total running time of the call to  $\text{CORESET}(P, k, \delta, \varepsilon)$  is in  $O(ndk \log(1/\delta))$ .

## E Additional tests

In the following section, we include additional tests, which were not in the main text due to space limitations.

### E.1 Tests of stand-alone imbalanced clustering

In the following section, we compare the proposed approximation and compression techniques to clustering methods, which aim to compute  $k$  clusters in one form or another, such as  $k$ -means [4] and Gaussian Mixtures [10].

We put tests for hierarchical clustering in Section E.2 since in the following section we consider clustering as a means to an end. In contrast, in Section E.2 the clustering is a subroutine of the algorithm considered.

**Labels.** All the methods used from the imbalanced-learn library [19] require labels/targets, which are supplied. In simple terms, we provide for each point to which class it belonged or was generated from.

**Approximation methods.** In this section, we consider the following three sets of approximation methods. In all the sets methods (i) and (ii) are identical, the name of the sets is defined by the other methods. I.e., **approximation** consists of methods that do not change the data set, **under-sample** consists

of methods that decrease the data set to obtain balanced sets, and **over-sample** consists of methods that increase the data set to obtain balanced sets.

**Approximation.**

- (i). The output of  $\text{APPROX}((P, w), k)$ , where  $P$  denotes the data and  $w : P \rightarrow \{1\}$  (each point is mapped to 1); see Definition 3. This is denoted by **APPROX**.
- (ii). The output of  $\text{APPROX}((P', w'), k)$ , where  $(P', w')$  is the output of a call to Algorithm 2 over the data. This is denoted by **Approx-on-coreset**.
- (iii). `kmeans++` [4] as implemented in `scikit-learn` [15], denoted by **Kmeans**.
- (iv). `kmeans++` as implemented by `RAPIDS` to utilize GPU [26], denoted by **Kmeans-Gpu**.
- (v). Our implementation of Algorithm 1, denoted by **BI-CRITERIA**.
- (vi). Gaussian mixture fitting as implemented in `scikit-learn`, with the initialization set to `kmeans++`, denoted by **gaussian**.

**Under-sample.**

From now on, when we state `kmeans++`, we refer to the `scikit-learn` implementation.

- (i). The previously stated **APPROX**.
- (ii). The previously stated **Approx-on-coreset**.
- (iii). `kmeans++` computed over **RandomUnderSampler** from the `imbalanced-learn` library [19], denoted by **RandomUnderSampler**.
- (iv–vi). `kmeans++` computed over **NearMiss**, which is based on [23], from the `imbalanced-learn` library. Since at the time of writing, **NearMiss** at the `imbalanced-learn` library had 3 versions, we utilized all the versions, denoted by **NearMiss-1**, **NearMiss-2**, **NearMiss-3**.

**Over-sample.**

- (i). The previously stated **APPROX**.
- (ii). The previously stated **Approx-on-coreset**.
- (iii). `kmeans++` computed over **RandomOverSampler** from the `imbalanced-learn` library, denoted by **RandomOverSampler**.
- (iv). `kmeans++` computed over the commonly used **SMOTE** [7], from the `imbalanced-learn` library. Denoted by **SMOTE**.
- (v). `kmeans++` computed over **KMeansSMOTE** [18], from the `imbalanced-learn` library. Denoted by **KMeansSMOTE**.
- (vi). `kmeans++` computed over **SMOTETomek**, which was presented at [5], from the `imbalanced-learn` library. Denoted by **SMOTETomek**. Note that this method utilizes a combination of over-sampling and under-sampling.

**Loss function.** In this section, for each dataset  $P \subset \mathbb{R}^2$  and its approximation  $C \subset \mathbb{R}^d$ , we use  $\ell(P, C)$  as the loss. For **gaussian**, for a dataset  $P \subset \mathbb{R}^2$ , for each set in the partition of  $P$  corresponding to the model (each point goes to its most probable Gaussian probability, ties broken arbitrarily), we set the center of the partition to be its mean. For readability, for each set generated, we divide all the losses by the loss of **APPROX**. It should be emphasized that our methods attempt to optimize the loss function from Definition 2, and not Definition 1.

**Synthetic.** In the following test, inspired by the motivation in Section 1.3, we consider the following data sets in  $\mathbb{R}^2$ , where we use different values for  $n$ , where we vary the size of  $n$  across the test, which are the union of 2 uniform samples from discs generated as follows.

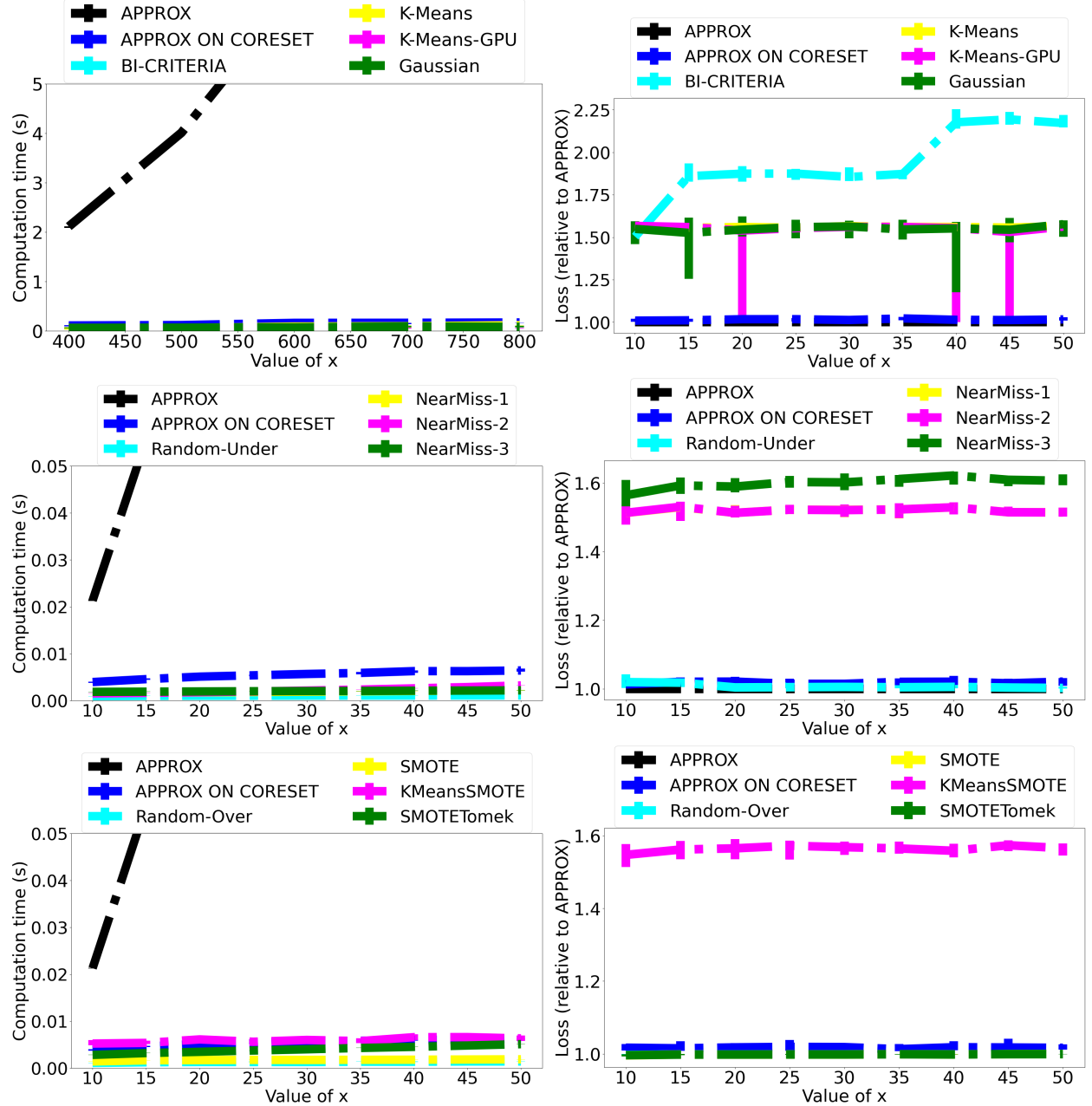
- (i). A sample of  $25n$  “inliers” points, where each point is chosen uniformly inside the unit disc.
- (ii). A sample of  $n$  “outliers” points, where each point is chosen uniformly inside a disc of radius 0.1, centered at  $(2.25, 0)$ .

We utilize the set  $\{5i\}_{i=1}^{10}$  as the values for  $x$ .

The results are presented in Figure 7. The test was repeated 40 times, and in all the figures the values presented are the medians across the tests, along with error bars that present the 25% and 75% percentiles.

The time taken to compute the optimal solution is rather large. Nonetheless, the time taken for **BI-CRITERIA** and **Approx-on-coreset** is significantly lower and is in line with the other method tested.

For the Approximation methods, **Approx-on-coreset** and **APPROX** have yielded a noticeable quality improvement compared to **Kmeans**. Notably, **Approx-on-coreset** has yielded an almost unnoticeable quality decrease compared to **APPROX**. While, **BI-CRITERIA**, due to returning more clusters yielded a higher loss than the other methods.



**Fig. 7.** The results for Section E.1. The left column corresponds to the running time of the method, and the right column corresponds to the loss of the methods, normalized for APPROX to be 1. The rows correspond, top to bottom, to **Top:** Approximation, **Middle:** Under-sample, and **Bottom:** Over-sample.

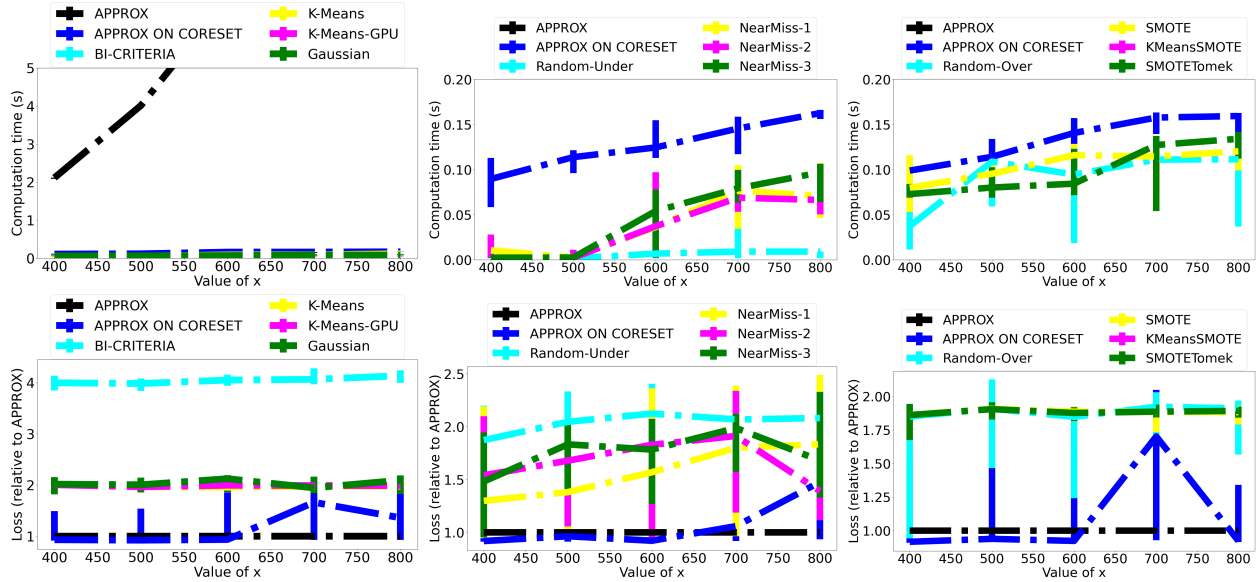
For the under-sampling methods, we obtained that `RandomUnderSampler` yielded essentially identical loss to our methods tested. This is rather logical since the method is given the labels, which are essentially the circles to which each point corresponds. Thus, by resampling the points such that in each circle we would have an equal number of points, `kmeans++` should classify the data correctly and yield low loss. Nonetheless, the other resampling methods have not yielded such improvement, and seem to have made the approximation worse than if they were not used at all.

For the over-sampling methods, we obtained that all the competing methods yielded essentially the same performance as `APPROX`, besides `KMeansSMOTE` which had a noticeably larger loss. Nonetheless, note that all the competing methods were given the labels of the points, and have increased the size of the data.

**Real world data.** In the following tests, we utilize the webpage data set, which was proposed in [8]. All the 34780 entries in this data have 300 features.

In each test iteration (done separately across the approximation sets) we uniformly, without repetitions, sample values from the data set. Specifically, we sample the values in the set  $\{100i\}_{i=4}^8$ ; chosen to not obtain error for most of the competing methods from the imbalanced-learn library.

The results are presented in Figure 8. The test was repeated 40 times, and in all the figures the values presented are the medians across the tests, along with error bars that present the 25% and 75% percentiles.



**Fig. 8.** The results for Section E.1. The top row corresponds to the running time of the method, and the bottom row corresponds to the loss of the methods, normalized for APPROX to be 1. The columns correspond, left to right, to **Left:** Approximation, **Middle:** Under-sample, and **Right:** Over-sample.

We have obtained parallel results to the test in Section E.1. To summarise, `Approx-on-coreset` and `APPROX` yielded the lowest loss and improved the results noticeably comparable to all the other methods (including the ones that are given the data set classes, while we do not use them). This is in contrast to Section E.1, where a few methods (that utilized labels) from the imbalanced-learn library [19] gave comparable results to `Approx-on-coreset` and `APPROX`.

Notably, our approximation over the coreset gave better results than the approximation over the entire data set. That is, `Approx-on-coreset` gave better quality than `APPROX`. A similar occurrence, where coreset improves the results, was previously noted in the coreset literature.

## E.2 Imbalanced clustering in hierarchical clustering

In the following section, we demonstrate the improvement of our proposed approximation and compression techniques to hierarchical clustering methods, specifically divisive hierarchical clustering.

Note that we do not address comparisons to agglomerative clustering, and use our method as a subroutine of divisive hierarchical clustering and not as a stand-alone one. As noted by a reviewer such a comparison has merit and we hope to add such one in future versions.

**Tests setup** In this section, we compare the following hierarchical clustering methods.

(i). Divisive clustering, where each split is done via **Approx-on-coreset**. That is, we compute 2 centers with **Approx-on-coreset**, and split the data by to which center each point is closest (euclidean distance). Denoted by

**Approx-on-coreset**.

(ii). Divisive clustering, where each split is done via **Kmeans**. That is, we compute 2 centers with **Kmeans**, and split the data by to which center each point is closest. Denoted by **Kmeans**.

(iii). Divisive clustering, where each split is done via **BI-CRITERIA**. That is, we compute centers with **BI-CRITERIA** and split the data by to which center each point is closest. Denoted by **BI-CRITERIA**.

(iv). Divisive clustering, where each split is done via by computing 2 centers by a call to

**Clustering.SpectralClustering** from skict-learn [25], and split the data by to which center each point is closest. Denoted by **spectral**.

(v). The output of the dePDDP divisive clustering from [2], denoted by **dePDDP**.

(vi). The output of the ward version of the agglomeration clustering [2], denoted by **ward**.

In all the tests the loss is the sum of the variances of the clusters computed.

In all the following tests we utilize 3 splits for all the methods, i.e., each method partitions the data to 8 sets.

**Synthetic data.** In the following test, we generate the data by a union of the following 3 sets, where we vary the size of  $x$  across the test.

(i). A sample of  $250x$  points from a circle in  $\mathbb{R}^2$ , where each point is chosen uniformly from the area inside the unit circle.

(ii). A sample of  $10x$  points from a circle in  $\mathbb{R}^2$ , where each point is chosen uniformly from the area inside a circle with center in  $(2.25, 0)$  and radius 0.1.

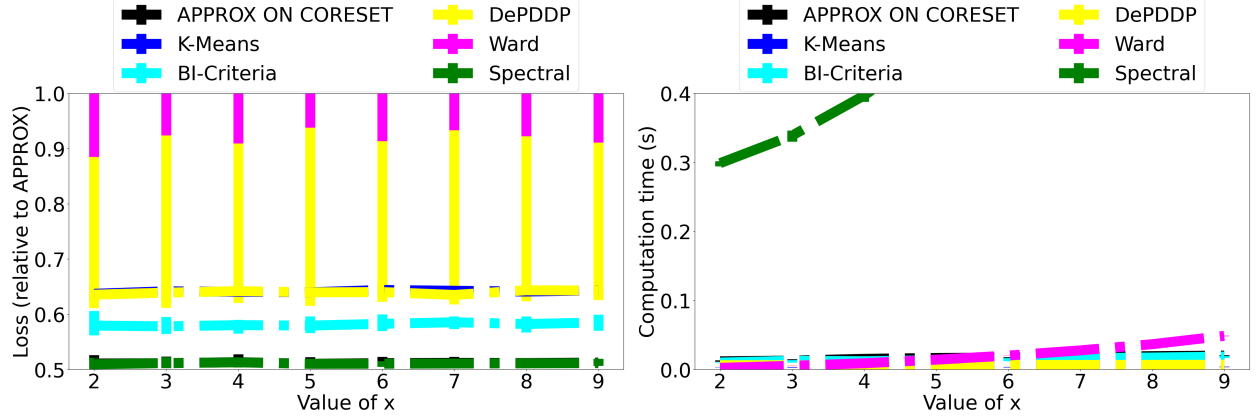
(ii). A sample of  $10x$  points from a circle in  $\mathbb{R}^2$ , where each point is chosen uniformly from the area inside a circle with center in  $(2.25, 2.25)$  and radius 0.1.

We utilize the set  $\{i\}_{i=1}^{10}$  as the values for  $x$ .

The results are presented in Figure 9. The test was repeated 100 times, and in all the figures the values presented are the medians across the tests, along with error bars that present the 25% and 75% percentiles.

As can be seen, the best results were for **spectral**, **Approx-on-coreset**, followed by **BI-CRITERIA** and **Kmeans**, **dePDDP** consecutively. Nonetheless, **spectral** and **Approx-on-coreset** yield noticeably lower loss. Moreover, observe that most of the competing methods have similar running times, except **spectral** which is notably slower, and **ward** which for the larger values of  $x$  becomes noticeably slower. It is expected that **ward** takes noticeably longer for larger values of  $x$ , due to its quadratic time complexity, while **Kmeans**, **Approx-on-coreset**, **Approx-on-coreset** all have semi-linear time complexity.

Hence, the only other method which has a similar quality to our method, i.e., **spectral**, is noticeably slower.

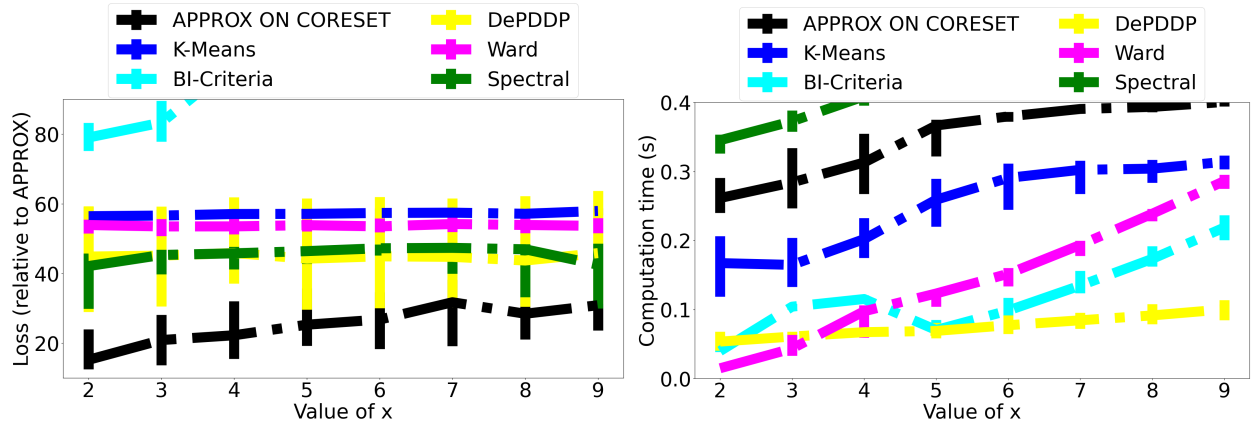


**Fig. 9.** The results for Section E.2. The left figure is the loss (sum of cluster variance) plot, and the right plot is the time plot.

**Real world data.** In the following tests, we utilize the webpage data set, which was proposed in [8]. All the 34780 entries in this data have 300 features.

In each test iteration (done separately across the tests) we uniformly, without repetitions, sample values from the data set. Specifically, we sample the values in the set  $\{250i\}_{i=2}^{10}$ ; chosen to not obtain error for most of the competing methods.

The results are presented in Figure 10. The test was repeated 100 times, and in all the figures the values presented are the medians across the tests, along with error bars that present the 25% and 75% percentiles.



**Fig. 10.** The results for Section E.2. The left figure is the loss (sum of cluster variance) plot, and the right plot is the time plot.

As can be seen, `Approx-on-coreset` has yielded the best results by a noticeable margin. While `Approx-on-coreset` has a noticeably slower running time than most other methods, as can be expected since the next faster method was `Kmeans`, where the computation of the  $k$ -means is utilized internally in `Approx-on-coreset`. Hence, we obtained the best results at the price of a longer running time. Nonetheless, note that our implementation is less optimized than the other competing methods, such as `Kmeans` which consists mostly of a call to `k-means++` implemented by `Scit-Learn`.