CORESETS FOR KERNEL CLUSTERING

Anonymous authors
Paper under double-blind review

ABSTRACT

We devise coresets for kernel k-MEANS with a general kernel, and use them to obtain new, more efficient, algorithms. Kernel k-MEANS has superior clustering capability compared to classical k-MEANS, particularly when clusters are separable non-linearly, but it also introduces significant computational challenges. We address this computational issue by constructing a coreset, which is a reduced dataset that accurately preserves the clustering costs.

Our main result is the first coreset for kernel k-MEANS with a general kernel, that has size independent of n, the number of input points; moreover, our coreset can be constructed in time near-linear in n. This result immediately implies new algorithms for kernel k-MEANS, such as a $(1+\epsilon)$-approximation in time near-linear in n, and a streaming algorithm using space and update time $\mathrm{poly}(k\epsilon^{-1}\log n)$.

We validate our coreset on various datasets with different kernels. Our coreset performs consistently well, achieving small errors while using very few points. We show that our coresets can speed up kernel k-MEANS++ (the kernelized version of the widely used k-MEANS++ algorithm), and we further use this faster kernel k-MEANS++ for spectral clustering. In both applications, we achieve up to 1000x speedup while the error is comparable to baselines that do not use coresets.

1 INTRODUCTION

We design the first coresets, and consequently new efficient algorithms, for kernel k-MEANS and related problems, like its generalization kernel (k,z)-CLUSTERING, under general kernels. The k-MEANS problem has proved to be fundamental for unsupervised learning in numerous application domains. Vanilla k-MEANS fails to capture sophisticated cluster structures, e.g., when the clusters are separable non-linearly, but this can be tackled by applying kernel methods (Schölkopf et al., 1998; Girolami, 2002). This has led to kernel k-MEANS, where data points are first mapped to a high-dimensional feature space (possibly implicitly via a kernel function), and then clustered in this richer space using a classical k-MEANS.

Formally, a *kernel* for a dataset X is a function $K : X \times X \to \mathbb{R}_+$ (intended to measure similarity between elements in X) that can be realized by inner products, i.e., there exist a Hilbert space $\mathcal{H}$ and a map $\varphi : X \to \mathcal{H}$ (called feature space and feature map) such that

$$\forall x, y \in X, \quad \langle \varphi(x), \varphi(y) \rangle = K(x,y). \tag{1}$$

In *kernel* k-MEANS, the input is a dataset X with weight function $w_X : X \to \mathbb{R}_+$ and a kernel function $K : X \times X \to \mathbb{R}_+$ as above, and the goal is to find a k-point center set $C \subseteq \mathcal{H}$ that minimizes the objective

$$\mathrm{cost}^{\varphi}(X,C) = \sum_{x \in X} w_X(x) \cdot \min_{c \in C} \|\varphi(x) - c\|^2. \tag{2}$$

(An equivalent formulation asks for a k-partitioning of X, keeping C implicit.)

This kernel version has superior clustering capability compared to classical k-MEANS (Zhang & Rudnicky, 2002; Kim et al., 2005), and has proved useful in different application domains, such as pattern recognition (Shawe-Taylor & Cristianini, 2004), natural language processing (Andrews & Fox, 2007), biology (Gönen & Margolin, 2014) and social networks (van Laarhoven & Marchiori, 2016). In fact, kernel k-MEANS is useful also for solving other clustering problems, such as normalized cut and spectral clustering (Dhillon et al., 2004; Ding et al., 2005).

Computational challenges. As observed in previous work (Girolami, 2002; Dhillon et al., 2004), the *kernel trick* can be applied to rewrite kernel k-MEANS using access only to the kernel $K(\cdot,\cdot)$ and without computing the very high-dimensional map φ explicitly. However, this approach has outstanding computational challenges (compared to classical k-MEANS), essentially because of the kernel trick. Consider the special case where $k = 1$ and the input is n unweighted points (i.e., 1-MEAN clustering). It is well known that the optimal center $c^\star$ has a closed form $c^\star := \frac{1}{n}\sum_{x\in X}\varphi(x)$. But the kernel trick requires $\Omega(n^2)$ accesses to K to evaluate $\mathrm{cost}^\varphi(X, c^\star)$,[1] while in the classical setting such evaluation needs only $O(n)$ distance computations.

This $\Omega(n^2)$ barrier can be bypassed by allowing $(1+\epsilon)$-approximation. In particular, let S be a uniform sample of $\mathrm{poly}(\epsilon^{-1})$ points from X, and let $\hat{c} := \frac{1}{|S|}\sum_{x\in S}\varphi(x)$ be its 1-MEAN; then with high probability, $\mathrm{cost}^\varphi(X,\hat{c}) \le (1+\epsilon)\,\mathrm{cost}^\varphi(X, c^\star)$ and evaluating $\mathrm{cost}^\varphi(X,\hat{c})$ takes only $\mathrm{poly}(\epsilon^{-1})n$ time. However, this uniform-sampling approach does not generally work for $k \ge 2$, because if the optimal clustering is highly imbalanced, a uniform sample is unlikely to include any point from a small cluster. Alternative approaches, such as dimension reduction, were also proposed to obtain efficient algorithms for kernel k-MEANS, but they too do not fully resolve the computational issue. We elaborate on these approaches in Section 1.2.

Our approach. To tackle this computational challenge, we adapt the notion of a coreset (Har-Peled & Mazumdar, 2004) to kernel k-MEANS. Informally, a coreset is a tiny reweighted subset of the original dataset on which the clustering cost is preserved within $(1\pm\epsilon)$ factor for all candidate centers $C \subseteq \mathcal{H}$. This notion has proved very successful for classical k-MEANS, e.g., to design efficient near-linear algorithms. In our context of kernel k-MEANS, a coreset of size s for an input of size $n = |X|$ has a huge advantage that its k optimal center points can all be represented as linear combinations of only s points in the feature space. Given these k optimal centers (as linear combinations), evaluating the distance between a point $\varphi(x)$ and such a center takes merely $O(s^2)$ time, instead of $O(n^2)$, and consequently the objective can be $(1+\epsilon)$-approximated in time $O(s^2kn)$. Moreover, it suffices to use k centers (again as linear combinations) that are $(1+\epsilon)$-approximately optimal for the coreset S.

In addition, coresets are very useful in dealing with massive datasets, since an offline construction of coresets usually generalizes to the streaming setting (Har-Peled & Mazumdar, 2004), distributed computing (Balcan et al., 2013) and dynamic algorithms (Henzinger & Kale, 2020) via the merge-and-reduce method (Har-Peled & Mazumdar, 2004), and existing (offline) algorithms can be efficiently applied to the coreset, instead of to the original dataset, with minor or no modifications.

1.1 OUR RESULTS

Our main result is the first coreset for kernel k-MEANS with a general kernel, that has size independent of the input size $n = |X|$; moreover, our coreset can be constructed in near-linear time for small k. (In fact, it generalizes to kernel (k,z)-CLUSTERING, see Section 2 for definitions.) Formally, an *ϵ-coreset* for kernel k-MEANS with respect to weighted dataset X and kernel function $K : X\times X \to \mathbb{R}_+$ is a weighted subset $S \subseteq X$, such that for every feature space $\mathcal{H}$ and feature map φ that realize K, as defined in (1),

$$\forall C \subseteq \mathcal{H}, |C| = k, \quad \mathrm{cost}^\varphi(S, C) \in (1\pm\epsilon)\cdot\mathrm{cost}^\varphi(X, C). \tag{3}$$

Previously, only a *weak* coreset was known for kernel k-MEANS (Feldman et al., 2007), meaning that the objective is preserved only for certain candidate centers (whereas (3) guarantees this for all centers), and that coreset works only for certain kernels (finite-dimensional). While we employ a similar approach, the technical differences make our bottom-line result much stronger.

Throughout, we assume an oracle access to K takes unit time, and therefore our stated running times also bound the number of accesses to K. We denote $\tilde{O}(f) = O(f\cdot\mathrm{polylog}\, f)$ to suppress logarithmic factors.

Theorem 1.1 (Informal version of Theorem 3.1)**.** *Given n-point weighted dataset X, oracle access to a kernel $K : X\times X\to\mathbb{R}_+$, integer $k \ge 1$ and $0 < \epsilon < 1$, one can construct in time $\tilde{O}(nk)$, a*

[1] In fact, evaluating $\|c^\star - \varphi(u)\|^2$ for a single point $u \in X$ already requires $\Theta(n^2)$ accesses, since $\|c^\star - \varphi(u)\|^2 = K(u,u) - \frac{2}{n}\sum_{x\in X}K(x,u) + \frac{1}{n^2}\sum_{x,y\in X}K(x,y)$.

reweighted subset $S \subseteq X$ of size $|S| = \mathrm{poly}(k\epsilon^{-1})$, that with high probability is an ϵ-coreset for kernel k-MEANS with respect to X and K.

We can employ our coreset to devise a $(1+\epsilon)$-approximation algorithm for kernel k-MEANS, that runs in time that is near-linear in n and parameterized by k. This is stated in Corollary 1.2, whose proof follows by solving k-MEANS on S optimally, using straightforward enumeration over all k-partitions of S. To the best of our knowledge, such a fast $(1+\epsilon)$-approximation for kernel k-MEANS was not known even for $k = 2$; for example, uniform sampling would fail in cases where the optimal clustering is very imbalanced, as mentioned earlier.

Corollary 1.2 (FPT-PTAS). *Given n-point weighted dataset X, oracle access to a kernel $K : X \times X \to \mathbb{R}_+$, integer $k \geq 1$ and $0 < \epsilon < 1$, one can compute in time $O(nk + k^{\mathrm{poly}(k\epsilon^{-1})})$, a center set C of k points, each represented as a linear combination of at most $\mathrm{poly}(k\epsilon^{-1})$ points from $\varphi(X)$, such that with high probability C is a $(1+\epsilon)$-approximation for kernel k-MEANS on X and K. In particular, given such C, one can find for each $x \in X$ its closest center in C in time $\mathrm{poly}(k\epsilon^{-1})$.*

In fact, for the purpose of finding near-optimal solutions, it already suffices to preserve the cost for centers coming from $\mathrm{span}(\varphi(X))$ (see Fact 2.1) which is an n-dimensional subspace. However, our definition of coreset in (3) is much stronger, in that the objective is preserved even for centers coming from a possibly infinite-dimensional feature space. This stronger guarantee ensures that the coreset is composable, and thus the standard merge-and-reduce method can be applied. In particular, our coreset implies the *first* streaming algorithm for kernel k-MEANS.

Corollary 1.3 (Streaming kernel k-MEANS). *There is a streaming algorithm that given a dataset X presented as a stream of n points, and oracle access to a kernel $K : X \times X \to \mathbb{R}_+$, constructs a reweighted subset $S \subseteq X$ of $\mathrm{poly}(k\epsilon^{-1})$ points using $\mathrm{poly}(k\epsilon^{-1} \log n)$ words of space and update time, such that with high probability S is an ϵ-coreset for k-MEANS with respect to X and K.*

Experiments and other applications. We validate the efficiency and accuracy of our coresets on various data sets with polynomial and Gaussian kernels, which are frequently-used kernels. For every dataset, kernel, and coreset-size that we test, our coreset performs consistently better than uniform sampling which serves as a baseline. In fact, our coreset achieves less than 10% error using only about 1000 points for every dataset.

We also showcase significant speedup to several applications that can be obtained using our coresets. Specifically, we adapt the widely used k-MEANS++ (Arthur & Vassilvitskii, 2007) to the kernel setting, and we compare the running time and accuracy of this kernelized k-MEANS++ with and without coresets. On a dataset of size 10^5, we observe more than $1000x$ speedup of k-MEANS++ when using coresets, while achieving a very similar error. Furthermore, this new efficient version of kernelized k-MEANS++ (based on coresets) is applied to solve spectral clustering, using the connection discoverd by Dhillon et al. (2004). Compared to the implementation provided by Scikit-learn (Pedregosa et al., 2011), our algorithm often achieves a better result and uses significantly less time. Hence, our coreset-based approach can potentially become the leading method for solving spectral clustering in practice.

1.2 COMPARISON TO PREVIOUS APPROACHES

The computational issue of kernel k-MEANS is an important research topic and has attracted significant attention. In the following, we compare our result with previous work that is representative of different possible approaches for the problem.

Uniform sampling of data points is a commonly used technique, which fits well in kernel clustering, because samples can be drawn without any access to the kernel. While the coresets that we use rely on sampling, we employ importance sampling, which is non-uniform by definition. Chitta et al. (2011) employs uniform sampling for kernel k-MEANS, but instead of solving kernel k-MEANS on a sample of data points directly (as we do), their method works in iterations, similarly to Lloyd's algorithm, that find a center set that is a linear combination of the sample. However, it has no worst-case guarantees on the error or on the running time, which could be much larger than $\tilde{O}(nk)$. Lastly, this method does not generalize easily to other sublinear settings, such as streaming (as our Corollary 1.3). Ren & Du (2020) analyze uniform sampling for k-MEANS in a Euclidean space,

which could be the kernel's feature space. In their analysis, the number of samples (and thus running time) crucially depends on the diameter of the dataset and on the optimal objective value, and is thus not bounded in the worse-case. This analysis builds on several earlier papers, for example Czumaj & Sohler (2007) achieve bounds of similar flavor for k-MEANS in general metric spaces.

Another common approach to speed up kernel k-MEANS is to approximate the kernel K using dimension-reduction techniques. In a seminal paper, Rahimi & Recht (2007) proposed a method that efficiently computes a low (namely, $O(\log n)$) dimensional feature map $\tilde{\varphi}$ that approximates φ (without computing φ explicitly), and this $\tilde{\varphi}$ can be used in downstream applications. Their method is based on designing random Fourier features, and works for a family of kernels that includes the Gaussian one, but not general kernels. This method was subsequently tailored to kernel k-MEANS by Chitta et al. (2012), and another followup work by Chen & Phillips (2017) established worst-case bounds for kernel k-MEANS with Gaussian kernels. Despite these promising advances, we are not aware of any work based on dimension reduction that can handle a general kernel function (as in our approach). In a sense, these dimension-reduction techniques are "orthogonal" to our data-reduction approach, and the two techniques can possibly be combined to yield even better results.

An alternative dimension-reduction approach is low-rank approximation of the kernel matrix K. Recent work by Musco & Musco (2017) and by Wang et al. (2019) presents algorithms based on Nyström approximation to compute a low-rank (namely, $O(k/\epsilon)$) approximation $\widetilde{K}$ to the kernel matrix K in time near-linear in n, such that the optimal kernel k-MEANS on $\widetilde{K}$ achieves $(1+\epsilon)$-approximation to that on the original kernel K. However, the low-rank $\widetilde{K}$ does not immediately imply efficient algorithms for kernel k-MEANS (for instance, Lloyd's algorithm still requires $\Theta(n^2)$ queries to $\widetilde{K}$ per iteration), while our coreset can be readily combined with off-the-shelf clustering algorithms.

2 PRELIMINARIES

Notation. A weighted set U is a finite set U associated with a weight function $w_U : U \to \mathbb{R}_+$. For such U, let $\|U\|_0$ be the number of distinct elements in it. For a weight function as above and a subset $S \subseteq U$, define $w_U(S) := \sum_{u\in S} w_U(u)$. For any other map $f : U \to V$ (not a weight function), we follow the standard definition $f(S) := \{f(x) : x \in S\}$. For an integer $t \geq 1$, let $[t] := \{1, \ldots, t\}$. For a real number x and an integer $i \geq 1$, let $\log^{(i)} x$ be the i-th iterated log of x, i.e., $\log^{(1)} x = \log x$ and for $i \geq 2$ let $\log^{(i)} x = \log(\log^{(i-1)} x)$.

Kernel functions. Let X be a set of n points. A function $K : X \times X \to \mathbb{R}_+$ is a kernel function if the $n \times n$ matrix M such that $M_{ij} = K(x_i, x_j)$ (where $x_i, x_j \in X$) is positive semi-definite. Since M is positive semi-definite, there exists a map φ from X to some Hilbert space $\mathcal{H}$, such that all $x, y \in X$ satisfy $K(x, y) = \langle \varphi(x), \varphi(y)\rangle$. This above (existence of a map φ of into $\mathcal{H}$) can be extended to infinite X, e.g., $X = \mathbb{R}^d$, by Mercer's Theorem. The distance between $x', y' \in \mathcal{H}$ is defined as $\mathrm{dist}(x', y') := \|x' - y'\| = \sqrt{\langle x' - y', x' - y'\rangle}$. Hence, the distance $\mathrm{dist}(\varphi(x), \varphi(y))$ for $x, y \in X$ can be represented using K as

$$\mathrm{dist}(\varphi(x), \varphi(y)) = \|\varphi(x) - \varphi(y)\| = \sqrt{K(x, x) + K(y, y) - 2K(x, y)}.$$

We refer to a survey by Ghojogh et al. (2021) for a more comprehensive introduction to kernel functions.

Kernel (k, z)-CLUSTERING. In the the kernel (k, z)-CLUSTERING problem, the input is a weighted data set X of n objects, a kernel function $K : X \times X \to \mathbb{R}_+$, an integer $k \geq 1$, and $z > 0$. The goal is to find a k-point center set $C \subseteq \mathcal{H}$ that minimizes the objective

$$\mathrm{cost}^{\varphi}_z(X, C) := \sum_{x\in X} w_X(x)(\mathrm{dist}(\varphi(x), C))^z, \tag{4}$$

where $\mathcal{H}$ is an induced Hilbert space of K and $\varphi : X \to \mathcal{H}$ is its feature map, and $\mathrm{dist}(\varphi(x), C) := \min_{c\in\mathcal{H}} \mathrm{dist}(\varphi(x), c) = \min_{c\in\mathcal{H}} \|\varphi(x) - c\|$. The case $z = 2$ clearly coincides with kernel k-MEANS whose objective is (2). The (non-kernel) (k, z)-CLUSTERING problem may be viewed as kernel (k, z)-CLUSTERING with kernel $K(x, y) = \langle x, y\rangle$ and identity feature map $\varphi(x) = x$.

While the feature map φ might not be unique, we show below that this kernel (k,z)-CLUSTERING is well defined, in the sense that the optimal value is independent of φ. The following two facts are standard and easy to prove.

Fact 2.1. *For every map φ into $\mathcal{H}$, there is an optimal solution $C^\star$ in which every center point $c \in C^\star$ lies inside $\mathrm{span}(\varphi(X))$, and is thus a linear combination of $\varphi(X)$.*

Corollary 2.2. *The optimal value of (4) can be represented as a function of kernel values $K(x,y)$, and is thus invariant of φ.*

ϵ-Coresets for kernel (k,z)-CLUSTERING. For $0 < \epsilon < 1$, an ϵ-coreset for kernel (k,z)-CLUSTERING on a weighted dataset X and a kernel function K is a reweighted subset $S \subseteq X$, such that for every Hilbert space $\mathcal{H}$ and map $\varphi : X \to \mathcal{H}$ satisfying (1), we have

$$\forall C \subseteq \mathcal{H}, |C| = k, \qquad \mathrm{cost}_z^\varphi(S,C) \in (1 \pm \epsilon) \cdot \mathrm{cost}_z^\varphi(X,C).$$

The case $z = 2$ clearly coincides with (3).

3 CORESETS FOR KERNEL (k,z)-CLUSTERING

Theorem 3.1. *Given n-point weighted dataset X, oracle access to a kernel $K : X \times X \to \mathbb{R}_+$, $z \geq 1$, integer $k \geq 1$, and $0 < \epsilon < 1$, one can construct in time $\tilde{O}(nk)$, a reweighted subset $S \subseteq X$ of size $\|S\|_0 = 2^{O(z)} \cdot \mathrm{poly}(k\epsilon^{-1})$, that with high constant probability is an ϵ-coreset for kernel (k,z)-CLUSTERING with respect to X and K.*

At a high level, our prove this theorem by employing recent constructions of coresets for (k,z)-CLUSTERING in Euclidean spaces, in which the coreset size is independent of the Euclidean dimension (Sohler & Woodruff, 2018; Feldman et al., 2020; Huang & Vishnoi, 2020; Braverman et al., 2021). However, these coresets are designed for finite-dimensional Euclidean spaces, and are thus not directly applicable to our feature space $\mathcal{H}$, which might have infinite dimension.

To employ these coreset constructions, we show in Lemma 3.2 that the data points in the feature space $\mathcal{H}$ embed into an $(n+1)$-dimensional (Euclidean) space, without any distortion to distances between data points and centers. This observation is similar to one previously made by Sohler & Woodruff (2018) for a different purpose. Due to this embedding, it suffices to construct a coreset for the limited setting where centers come only from an $(n+1)$-dimensional space (Corollary 3.3).

Lemma 3.2. *Let $\mathcal{H}$ be a Hilbert space and let $X \subseteq \mathcal{H}$ be a subset of n points. Then there exists a map $f : \mathcal{H} \to \mathbb{R}^{n+1}$ such that*

$$\forall x \in X, c \in \mathcal{H}, \qquad \|x - c\| = \|f(x) - f(c)\|.$$

Proof. Let $\mathcal{S} = \mathrm{span}(X)$. Then every point $c \in \mathcal{H}$ can be written (uniquely) as $c = c^\parallel + c^\perp$, where $c^\parallel \in \mathcal{S}$ and $c^\perp$ is orthogonal to $\mathcal{S}$. Thus, $\|c\|^2 = \|c^\parallel\|^2 + \|c^\perp\|^2$. Note that for all $x \in X$, we have $x^\perp = 0$. Now, for every $c \in \mathcal{H}$, let $f(c) := (c^\parallel \,;\, \|c^\perp\|)$, where we interpret $x^\parallel$ as an n-dimensional vector. Then for all $x \in X$ and $c \in \mathcal{H}$,

$$\|x - c\|^2 = \|x^\parallel - c^\parallel\|^2 + \|x^\perp - c^\perp\|^2 = \|x^\parallel - c^\parallel\|^2 + \|c^\perp\|^2 = \|f(x) - f(c)\|^2.$$

The claim follows. □

Corollary 3.3. *Consider n-point weighted dataset X, kernel function $K : X \times X \to \mathbb{R}_+$, $z \geq 1$, integer $k \geq 1$, and $0 < \epsilon < 1$. Suppose that a reweighted subset $S \subseteq X$ satisfies that for every $\varphi : X \to \mathbb{R}^{n+1}$ such that for all $x, y \in X$, $\langle \varphi(x), \varphi(y) \rangle = K(x,y)$, the following holds*

$$\forall C \subseteq \mathbb{R}^{n+1}, |C| = k, \qquad \mathrm{cost}_z^\varphi(S,C) \in (1 \pm \epsilon) \cdot \mathrm{cost}_z^\varphi(X,C).$$

Then S is an ϵ-coreset for kernel (k,z)-CLUSTERING with respect to X and kernel K.

Proof. To verify that S is a coreset with respect to X and K, consider some feature space $\mathcal{H}'$ and feature map φ' be induced by K. Apply Lemma 3.2 to obtain $f : \mathcal{H}' \to \mathbb{R}^{n+1}$, then for all $C \subseteq \mathcal{H}'$, $|C| = k$, we have $\forall Q \subseteq X, \mathrm{cost}_z^{\varphi'}(Q,C) = \mathrm{cost}_z^{f \circ \varphi'}(Q, f(C))$, and using the promise about S with $\varphi = f \circ \varphi'$,

$$\mathrm{cost}_z^{\varphi'}(S,C) = \mathrm{cost}_z^{\varphi}(S, f(C)) \in (1 \pm \epsilon) \cdot \mathrm{cost}^{\varphi}(X, f(C)) = (1 \pm \epsilon) \cdot \mathrm{cost}^{\varphi'}(X,C).$$

Thus, S is indeed a coreset with respect to X and K. □

Algorithm 1 Constructing ϵ-coreset for kernel (k, z)-CLUSTERING on data set X with kernel K

1: let $X_0 \leftarrow X$, $i \leftarrow 0$
2: **repeat**
3: let $i \leftarrow i + 1$ and $\epsilon_i \leftarrow \epsilon/(\log^{(i)} \|X\|_0)^{1/4}$
4: $X_i \leftarrow$ IMPORTANCE-SAMPLING(X_{i-1}, ϵ_i)
5: **until** $\|X_i\|_0$ does not decrease compared with $\|X_{i-1}\|_0$
6: return X_i

Another issue is that some of the existing algorithms, such as Sohler & Woodruff (2018); Feldman et al. (2020), require the input to be an explicit representations of points in $\varphi(X)$, which is very expensive to compute. Fortunately, the importane-sampling-based algorithms of Huang & Vishnoi (2020) and Braverman et al. (2021) are oblivious to the representation of φ, and only rely on a distance oracle that evaluates $\|\varphi(x) - \varphi(y)\| = \sqrt{K(x,x) + K(y,y) - 2K(x,y)}$ for data points $x, y \in X$. Now, by Corollary 3.3, executing these algorithms without any modifications (except for plugging in the distance oracle defined by kernel K) yields the desired coreset for kernel (k, z)-CLUSTERING. We choose to use the coreset construction of Braverman et al. (2021), which is arguably simpler. We now recall its statement for completeness.

Theorem 3.4 (Braverman et al. (2021))**.** *Given n-point weighted dataset $X \subset \mathbb{R}^m$ for some integer m, together with $z \geq 1$, integer $k \geq 1$, and $0 < \epsilon < 1$, one can construct in time $\tilde{O}(nk)$, a reweighted subset $S \subseteq X$ of size $\|S\|_0 = \tilde{O}(\epsilon^{-4} 2^{2z} k^2)$, that with high constant probability is an ϵ-coreset for (k, z)-CLUSTERING with respect to X.*

Proof of Theorem 3.1. It follows immediately by combining Corollary 3.3 and Theorem 3.4. □

3.1 DESCRIPTION OF CORESET CONSTRUCTION ALGORITHMS

Next, we present our full algorithm in Algorithm 1 (which depends on the subroutines defined in Algorithm 2 and 3). While it essentially tailors previous work to our kernel clustering setting, we provide full details for completeness. The following notation is needed. For a subset $C \subseteq \mathcal{H}$ and data point $x \in X$, define $\mathrm{NN}_C(x) := \arg\min\{\mathrm{dist}(\varphi(x), y) : y \in C\}$ as the nearest neighbor of x in C with respect to the distances in the feature space (breaking ties arbitrarily). Thus $\mathrm{NN}_C(\cdot)$ defines a $|C|$-partition of X, and the cluster that x belongs to (with respect to C) is denoted $C(x) := \{x' \in X : \mathrm{NN}_C(x') = \mathrm{NN}_C(x)\}$.

Algorithm 1 is the main procedure for constructing the coreset, and its loop iteratively executes another importance-sampling-based coreset construction (Algorithm 2). Informally, each invocation of IMPORTANCE-SAMPLING constructs a coreset X_i from the current coreset X_{i-1}, to reduce the number of distinct elements in X_{i-1} to roughly $\log \|X_{i-1}\|_0$. The procedure ends when such size reduction cannot be done any more, at which point the size of the coreset reaches the bound in Theorem 3.4, which is independent of n.

In fact, subroutine IMPORTANCE-SAMPLING already constructs a coreset, albeit it is of a worse size that depends on $\log \|X\|_0$. This subroutine is based on the well-known importance sampling approach that was proposed and improved in a series of works (cf. Langberg & Schulman (2010); Feldman & Langberg (2011); Feldman et al. (2020)). Its first step is to compute an importance score σ_x for every data point $x \in X$ (lines 1–2), and then draw independent samples from X with probability proportional to σ_x (lines 3–4). The final coreset is formed by reweighting the sampled points (lines 5–6). Roughly speaking, the importance score σ_x measures the relative contribution of x to the objective function in the worst-case, which here means the maximum over all choices of the center set. It can be computed from an $O(\log k)$-approximate solution for kernel (k, z)-CLUSTERING on X, which is generated by the D^z-sampling subroutine (Algorithm 3), a natural generalization of the D^2-sampling introduced by Arthur & Vassilvitskii (2007).

We stress that our algorithm description uses the feature vectors $\varphi(x)$ for clarity of exposition. These vectors do not have to be provided explicitly, because only distances between them are required, and thus all steps can be easily implemented using the kernel trick, and the total time (and number of accesses to the kernel function K) is only $\tilde{O}(nk)$.

Algorithm 2 IMPORTANCE-SAMPLING(X, ϵ)

let $C^\star \leftarrow D^z$-SAMPLING(X)
for each $x \in X$, let $\sigma_x \leftarrow w_X(x) \cdot \left(\frac{(\mathrm{dist}(x,C^\star))^z}{\mathrm{cost}^\varphi_z(X,C^\star)} + \frac{1}{w_X(C^\star(x))} \right)$
for each $x \in X$, let $p_x \leftarrow \frac{\sigma_x}{\sum_{y \in X} \sigma_y}$
draw $N = O(\epsilon^{-4} 2^{2z} z k^2 \log^2 k \log \|X\|_0)$ i.i.d. samples from X, using probabilities $(p_x)_{x \in X}$
let D be the sampled set, and for each $x \in D$ let $w_D(x) \leftarrow \frac{w_X(x)}{p_x N}$
return weighted set D

Algorithm 3 D^z-SAMPLING(X)

▷ the feature vectors $\varphi(\cdot)$ are mentioned for clarity and not needed for implementation
let x be a uniform random point from X, and initialize $C \leftarrow \{\varphi(x)\}$
for $i = 1, \ldots, k-1$ **do**
    draw one sample $x \in X$, using probabilities $w_X(x) \cdot \frac{(\mathrm{dist}(\varphi(x),C))^z}{\mathrm{cost}^\varphi_z(X,C)}$
    let $C \leftarrow C \cup \{\varphi(x)\}$
end for
return C

4 EXPERIMENTS

We validate the empirical performance of our coreset for kernel k-MEANS against various datasets, and show that our coresets can significantly speedup a kernelized version of the widely used k-MEANS++ (Arthur & Vassilvitskii, 2007). In addition, we apply this new coreset-based kernelized k-MEANS++ to spectral clustering (via a reduction devised by Dhillon et al. (2004)), showing that it outperforms the well-known Scikit-learn solver in both running time and objective value.

Experimental setup. Our experiments are conducted on standard clustering datasets that consist of vectors in $\mathbb{R}^d$, and we use the RBF kernel (radial basis function kernel, also known as Gaussian kernel) and polynomial kernels as kernel functions. An RBF kernel K_G is of the form $K_G(x,y) := \exp\left(-\frac{\|x-y\|}{2\sigma^2}\right)$, where $\sigma > 0$ is a parameter, and a polynomial kernel K_P is of the form $K_P(x,y) := (\langle x,y \rangle + c)^d$ where c and d are parameters. Table 1 summarizes the specifications of datasets and our choice of the parameters for the kernel function. We note that the parameters are dataset-dependent, and that for Twitter and Census1990 dataset we subsample to 10^5 points since otherwise it takes too long to run for some of our inefficient baselines. Unless otherwise specified, we use a typical value $k = 5$ for the number of clusters. All experiments are conducted on a PC with Intel Core i7 CPU and 16 GB memory, and algorithms are implemented using C++.

4.1 SIZE AND EMPIRICAL ERROR TRADEOFF

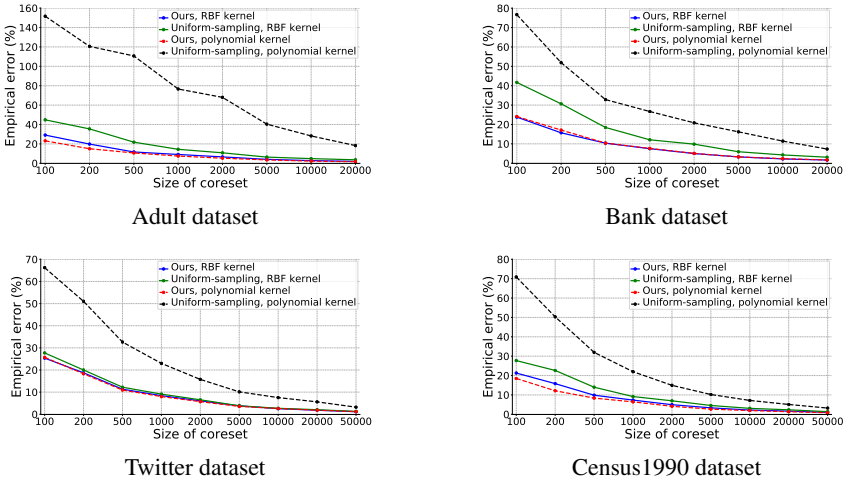
Our first experiment evaluates the empirical error versus coreset size. In our coreset implementation, we simplify the construction in Algorithm 1 by running the importance sampling step only once instead of running it iteratively, and it turns out this simplification still achieves excellent performance. As in many previous implementations, instead of setting ϵ and solving for the number of samples N in the IMPORTANCE-SAMPLING procedure (Algorithm 2), we simply set N as a parameter to directly control the coreset size. We construct the coreset with this N and evaluate its error by drawing 500 random center sets $\mathcal{C}$ (each consisting of k points) from the data set, and evaluate the maximum empirical error, defined as

$$\hat{\epsilon} := \max_{C \in \mathcal{C}} \frac{|\mathrm{cost}^\varphi(S,C) - \mathrm{cost}^\varphi(X,C)|}{\mathrm{cost}^\varphi(X,C)}. \tag{5}$$

This empirical error is measured similarly to the definition of coreset, except that it is performed on a sample of center sets. To make the measurement stable, the empirical error is evaluated independently 100 times and the average is reported.

Table 1: Specifications of datasets

dataset	size	RBF kernel param.	poly. kernel param.
Twitter (Chan et al., 2018)	21040936	$\sigma = 50$	$c = 0, d = 4$
Census1990 (Meek et al., 1990)	2458284	$\sigma = 100$	$c = 0, d = 4$
Adult (Dua & Graff, 2017)	48842	$\sigma = 200000$	$c = 0, d = 2$
Bank (Moro et al., 2014)	41188	$\sigma = 500$	$c = 0, d = 4$



Adult dataset | Bank dataset

Twitter dataset | Census1990 dataset

Figure 1: Tradeoffs between coreset size and empirical error.

The tradeoff of the coreset size versus empirical error is shown in Figure 1, where we also compare with a baseline that constructs the coreset using uniform sampling. These experiments show that our coreset performs consistently well on all the datasets and kernels. Furthermore, our coreset admits a similar error curve regardless of dataset and kernel function – for example, one can get within 10% error using a coreset of only 1000 points, – which is perfectly justified by our theory that the size of ϵ-coreset only depends on ϵ and k. Comparing with the uniform-sampling baseline, our coreset generally has superior performance, especially when the coreset size is small. We also observe that the uniform sampling suffers a larger variance compared with our coreset.

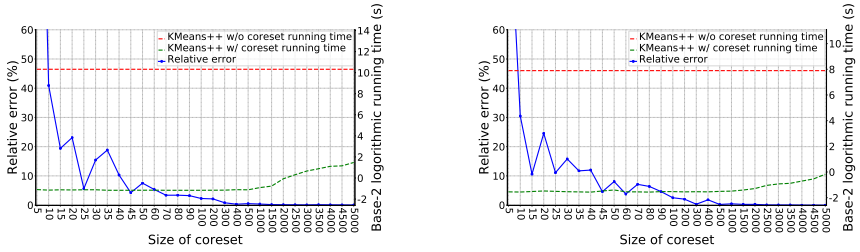
4.2 Speeding Up Kernelized k-Means++

k-Means++ (Arthur & Vassilvitskii, 2007) is a widely-used algorithm for k-Means, and it could easily be adopted to solve kernel k-Means by using the kernel trick, however as mentioned earlier this would take $\Omega(n^2)$ time. We use our coreset to speedup this kernelized k-Means++ algorithm, by first computing the coreset and then running kernelized k-Means++ on the coreest; this yields an implementation of kernelized k-Means++ whose running time is near-linear (in n).

In Figure 2, we demonstrate the running time and the error achieved by kernelized k-Means++ with and without coresets, experimented with varying coreset sizes. We measure the relative error of our coreset-based kernelized k-Means++ by comparing the objective value it achieves with that of vanilla (i.e., without coreset) kernelized k-Means++. These experiments show that the error decreases significantly as the coreset size increases, and it stabilizes around size $N = 100$, achieving merely $< 5\%$ error. Naturally, the running time of our coreset-based approach increases with the coreset size, but even the slowest one is still several orders of magnitude faster than vanilla kernelized k-Means++.

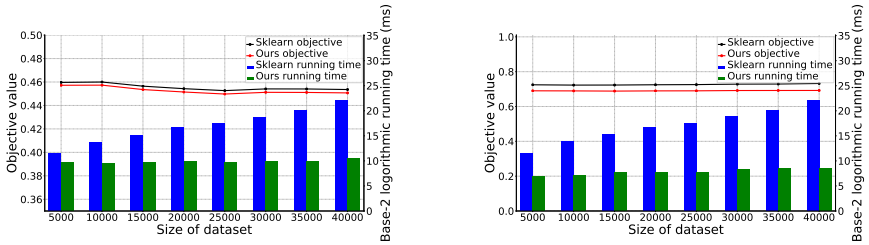
4.3 Speeding Up Spectral Clustering

In the spectral clustering problem, the input is a set of n objects X and an $n \times n$ similarity matrix A that measures the similarity between every pair of elements in X, and the goal is to find a k-



RBF kernel Polynomial kernel

Figure 2: Speedup of kernelized k-MEANS++ using our coreset. This experiment is conducted on the Twitter dataset with RBF and polynomial kernels. We run each algorithm 10 times, report the average running time and the minimum objective value (in relative-error evaluation).



RBF kernel Polynomial kernel

Figure 3: Speedup of spectral clustering using coreset-based kernelized k-MEANS++, with coreset size $N = 2000$. Similarly to Figure 2, we run each algorithm 10 times, report the average running time and the minimum objective value.

partition of X such that a certain objective function with respect to A is minimized. Dhillon et al. (2004) shows a way to write spectral clustering as a (weighted) kernel k-MEANS problem, which is eventually used to produce a spectral clustering. Specifically, let D be an $n \times n$ diagonal matrix such that $D_{ii} = \sum_{j \in [n]} A_{ij}$. Then according to Dhillon et al. (2004), spectral clustering can be written as a weighted kernel k-MEANS problem with weights $w_i := D_{ii}$ and kernel function $K := D^{-1}AD^{-1}$, provided that A is positive semidefinite (which could be viewed as a kernel). We use this reduction, and plug in the abovementioned coreset-based kernelized k-MEANS++ as the solver for kernel k-MEANS. We experiment on the subsampled Twitter dataset with varying number of points, and we use the polynomial and RBF kernels as the similarity matrix A.

However, we would need $\Theta(n^2)$ time if we evaluate D_{ii} naively. To resolve this issue, we draw a uniform sample S from $[n]$, and use $\hat{D}_{ii} := \frac{n}{|S|} \sum_{j \in S} A_{ij}$ as an estimate for D_{ii}. The accuracy of $\hat{D}$ is justified by previous work on kernel density estimation (Joshi et al., 2011, Theorem 5.2), and for our application we simply set $|S| = 1000$, which achieves good accuracy.

We compare our implementation with the spectral clustering solver from the well-known Scikit-learn library (Pedregosa et al., 2011) as a baseline. The experimental results, reported in Figure 3, show that our approach has more than 1000x of speedup already for moderately large datasets ($n = 40000$). This difference might be partially caused by efficiency issues of the Python language used for the Scikit-learn implementation (recall that our implementation is in C++), but we actually see that the asymptotic growth of our algorithm's running time is also much better than that of the Scikit-learn baseline. This suggests that our improvement in running time is fundamental, and not only due to the programming language. We also observe that our approach yields better objective values than Scikit-learn. One possible reason is that Scikit-learn might be conservative in using more iterations to gain better accuracy, because of the expensive computational cost that we do not suffer.

REFERENCES

Nicholas O Andrews and Edward A Fox. Recent developments in document clustering. Technical Report TR-07-35, Department of Computer Science, Virginia Polytechnic Institute & State University, 2007.

David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In *SODA*, pp. 1027–1035. SIAM, 2007.

Maria-Florina Balcan, Steven Ehrlich, and Yingyu Liang. Distributed k-means and k-median clustering on general communication topologies. In *NIPS*, pp. 1995–2003, 2013.

Vladimir Braverman, Shaofeng H.-C. Jiang, Robert Krauthgamer, and Xuan Wu. Coresets for clustering in excluded-minor graphs and beyond. In *SODA*, pp. 2679–2696. SIAM, 2021.

T-H. Hubert Chan, Arnaud Guerquin, and Mauro Sozio. Twitter data set, 2018. URL https://github.com/fe6Bc5R4JvLkFkSeExHM/k-center.

Di Chen and Jeff M. Phillips. Relative error embeddings of the gaussian kernel distance. In *ALT*, volume 76 of *Proceedings of Machine Learning Research*, pp. 560–576. PMLR, 2017.

Radha Chitta, Rong Jin, Timothy C. Havens, and Anil K. Jain. Approximate kernel k-means: Solution to large scale kernel clustering. In *KDD*, pp. 895–903. ACM, 2011.

Radha Chitta, Rong Jin, and Anil K. Jain. Efficient kernel clustering using random Fourier features. In *ICDM*, pp. 161–170. IEEE Computer Society, 2012.

Artur Czumaj and Christian Sohler. Sublinear-time approximation algorithms for clustering via random sampling. *Random Struct. Algorithms*, 30(1-2):226–256, 2007. doi: 10.1002/rsa.20157.

Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. Kernel k-means: spectral clustering and normalized cuts. In *KDD*, pp. 551–556. ACM, 2004.

Chris H. Q. Ding, Xiaofeng He, and Horst D. Simon. Nonnegative lagrangian relaxation of *K*-means and spectral clustering. In *ECML*, volume 3720 of *Lecture Notes in Computer Science*, pp. 530–538. Springer, 2005.

Dheeru Dua and Casey Graff. UCI machine learning repository, adult dataset, 2017. URL https://archive.ics.uci.edu/ml/datasets/adult.

Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *STOC*, pp. 569–578. ACM, 2011.

Dan Feldman, Morteza Monemizadeh, and Christian Sohler. A PTAS for k-means clustering based on weak coresets. In *SoCG*, pp. 11–18. ACM, 2007.

Dan Feldman, Melanie Schmidt, and Christian Sohler. Turning big data into tiny data: Constant-size coresets for k-means, PCA, and projective clustering. *SIAM J. Comput.*, 49(3):601–657, 2020.

Benyamin Ghojogh, Ali Ghodsi, Fakhri Karray, and Mark Crowley. Reproducing kernel Hilbert space, Mercer's theorem, eigenfunctions, Nyström method, and use of kernels in machine learning: Tutorial and survey. *CoRR*, abs/2106.08443, 2021.

Mark A. Girolami. Mercer kernel-based clustering in feature space. *IEEE Trans. Neural Networks*, 13(3):780–784, 2002.

Mehmet Gönen and Adam A. Margolin. Localized data fusion for kernel k-means clustering with application to cancer biology. In *NIPS*, pp. 1305–1313, 2014.

Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *STOC*, pp. 291–300. ACM, 2004.

Monika Henzinger and Sagar Kale. Fully-dynamic coresets. In *ESA*, volume 173 of *LIPIcs*, pp. 57:1–57:21. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

Lingxiao Huang and Nisheeth K. Vishnoi. Coresets for clustering in Euclidean spaces: Importance sampling is nearly optimal. In *STOC*, pp. 1416–1429. ACM, 2020.

Sarang C. Joshi, Raj Varma Kommaraju, Jeff M. Phillips, and Suresh Venkatasubramanian. Comparing distributions and shapes using the kernel distance. In *SoCG*, pp. 47–56. ACM, 2011.

Dae-Won Kim, Ki Young Lee, Doheon Lee, and Kwang Hyung Lee. Evaluation of the performance of clustering algorithms in kernel-induced feature space. *Pattern Recognit.*, 38(4):607–611, 2005.

Michael Langberg and Leonard J. Schulman. Universal epsilon-approximators for integrals. In *SODA*, pp. 598–607. SIAM, 2010.

Chris Meek, Bo Thiesson, and David Heckerman. UCI machine learning repository, census1990 dataset, 1990. URL http://archive.ics.uci.edu/ml/datasets/US+Census+Data+(1990).

S. Moro, P. Cortez, and P. Rita. UCI machine learning repository, bank dataset, 2014. URL https://archive.ics.uci.edu/ml/datasets/Bank+Marketing.

Cameron Musco and Christopher Musco. Recursive sampling for the Nystrom method. In *NIPS*, pp. 3833–3845, 2017.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *NIPS*, pp. 1177–1184. Curran Associates, Inc., 2007.

Yuanhang Ren and Ye Du. Uniform and non-uniform sampling methods for sub-linear time k-means clustering. In *ICPR*, pp. 7775–7781. IEEE, 2020.

Bernhard Schölkopf, Alexander J. Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Comput.*, 10(5):1299–1319, 1998.

John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

Christian Sohler and David P. Woodruff. Strong coresets for k-median and subspace approximation: Goodbye dimension. In *FOCS*, pp. 802–813. IEEE Computer Society, 2018.

Twan van Laarhoven and Elena Marchiori. Local network community detection with continuous optimization of conductance and weighted kernel k-means. *J. Mach. Learn. Res.*, 17:147:1–147:28, 2016.

Shusen Wang, Alex Gittens, and Michael W. Mahoney. Scalable kernel k-means clustering with Nyström approximation: Relative-error bounds. *J. Mach. Learn. Res.*, 20:12:1–12:49, 2019. URL http://jmlr.org/papers/v20/17-517.html.

Rong Zhang and Alexander I. Rudnicky. A large scale clustering scheme for kernel k-means. In *ICPR (4)*, pp. 289–292. IEEE Computer Society, 2002.