MINI-BATCH KERNEL k-MEANS

Anonymous authors
Paper under double-blind review

ABSTRACT

We present the first mini-batch kernel k-means algorithm, offering an order of magnitude improvement in running time compared to the full batch algorithm. A single iteration of our algorithm takes $\widetilde{O}(kb^2)$ time, significantly faster than the $O(n^2)$ time required by the full batch kernel k-means, where n is the dataset size and b is the batch size. Extensive experiments demonstrate that our algorithm consistently achieves a 10-100x speedup with minimal loss in quality, addressing the slow runtime that has limited kernel k-means adoption in practice. We further complement these results with a theoretical analysis under an early stopping condition, proving that with a batch size of $\widetilde{\Omega}(\max\left\{\gamma^4, \gamma^2\right\} \cdot k\epsilon^{-2})$, the algorithm terminates in $O(\gamma^2/\epsilon)$ iterations with high probability, where γ bounds the norm of points in feature space and ϵ is a termination threshold. Our analysis holds for any reasonable center initialization, and when using k-means++ initialization, the algorithm achieves an approximation ratio of $O(\log k)$ in expectation. For normalized kernels, such as Gaussian or Laplacian it holds that $\gamma = 1$. Taking $\epsilon = O(1)$ and $b = \Theta(k \log n)$, the algorithm terminates in $O(1)$ iterations, with each iteration running in $\widetilde{O}(k^3)$ time.

1 INTRODUCTION

Mini-batch methods are among the most successful tools for handling huge datasets for machine learning. Notable examples include Stochastic Gradient Descent (SGD) and mini-batch k-means (Sculley, 2010). Mini-batch k-means is one of the most popular clustering algorithms used in practice (Pedregosa et al., 2011).

While k-means is widely used due to it's simplicity and fast running time, it requires the data to be *linearly separable* to achieve meaningful clustering. Unfortunately, many real-world datasets do not have this property. One way to overcome this problem is to project the data into a high, even *infinite*, dimensional space (where it is hopefully linearly separable) and run k-means on the projected data using the "kernel-trick".

Kernel k-means achieves significantly better clustering compared to k-means in practice. However, its running time is considerably slower. Surprisingly, prior to our work there was no attempt to speed up kernel k-means using a mini-batch approach.

Problem statement We are given an input (dataset), $X = \{x_i\}_{i=1}^n$, of size n and a parameter k representing the number of clusters. A kernel for X is a function $K : X \times X \to \mathbb{R}$ that can be realized by inner products. That is, there exists a Hilbert space $\mathcal{H}$ and a map $\phi : X \to \mathcal{H}$ such that $\forall x, y \in X, \langle \phi(x), \phi(y) \rangle = K(x, y)$. We call $\mathcal{H}$ the *feature space* and ϕ the *feature map*.

In kernel k-means the input is a dataset X and a kernel function K as above. Our goal is to find a set $\mathcal{C}$ of k centers (elements in $\mathcal{H}$) such that the following goal function is minimized:

$$\frac{1}{n} \sum_{x \in X} \min_{c \in \mathcal{C}} \|c - \phi(x)\|^2.$$

Equivalently we may ask for a partition of X into k parts, keeping $\mathcal{C}$ implicit.[1]

[1] A common variant of the above is when every $x \in X$ is assigned a weight $w_x \in \mathbb{R}^+$ and we aim to minimize $\sum_{x \in X} w_x \cdot \min_{c \in \mathcal{C}} \|c - \phi(x)\|^2$. Everything that follows, including our results, can be easily generalized to the weighted case. We present the unweighted case to improve readability.

Lloyd's algorithm The most popular algorithm for (non kernel) k-means is Lloyd's algorithm, often referred to as the k-means algorithm (Lloyd, 1982). It works by randomly initializing a set of k centers and performing the following two steps: (1) Assign every point in X to the center closest to it. (2) Update every center to be the mean of the points assigned to it. The algorithm terminates when no point is reassigned to a new center. This algorithm is extremely fast in practice but has a worst-case exponential running time (Arthur & Vassilvitskii, 2006; Vattani, 2011).

Mini-batch k-means To update the centers, Lloyd's algorithm must go over the entire input at every iteration. This can be computationally expensive when the input data is extremely large. To tackle this, the mini-batch k-means method was introduced by Sculley (2010). It is similar to Lloyd's algorithm except that steps (1) and (2) are performed on a batch of b elements sampled uniformly at random with repetitions, and in step (2) the centers are updated slightly differently. Specifically, every center is updated to be the weighted average of its current value and the mean of the points (in the batch) assigned to it. The parameter by which we weigh these values is called the *learning rate*, and its value differs between centers and iterations. The larger the learning rate, the more a center will drift towards the new batch cluster mean.

Lloyd's algorithm in feature space Implementing Lloyd's algorithm in feature space is challenging as we cannot explicitly keep the set of centers $\mathcal{C}$. Luckily, we can use the kernel function together with the fact that centers are always set to be the mean of cluster points to compute the distance from any point $x \in X$ in feature space to any center $c = \frac{1}{|A|}\sum_{y\in A}\phi(y)$ as follows:

$$\begin{aligned}\|\phi(x) - c\|^2 &= \langle \phi(x) - c, \phi(x) - c\rangle = \langle \phi(x), \phi(x)\rangle - 2\langle \phi(x), c\rangle + \langle c, c\rangle \\ &= \langle \phi(x), \phi(x)\rangle - 2\langle \phi(x), \frac{1}{|A|}\sum_{y\in A}\phi(y)\rangle + \langle \frac{1}{|A|}\sum_{y\in A}\phi(y), \frac{1}{|A|}\sum_{y\in A}\phi(y)\rangle,\end{aligned}$$

where A can be any subset of the input X. While the above can be computed using only kernel evaluations, it makes the update step significantly more costly than standard k-means. Specifically, the complexity of the above may be quadratic in n (Dhillon et al., 2004).

Mini-batch kernel k-means Applying the mini-batch approach for kernel k-means is even more difficult because the assumption that cluster centers are always the mean of some subset of X in feature space no longer holds.

In Section 4 we first derive a recursive expression that allows us to compute the distances of all points to current cluster centers (in feature space). Using a simple dynamic programming approach that maintains the inner products between the data and centers in feature space, we achieve a running time of $O(n(b+k))$ per iteration compared to $O(n^2)$ for the full-batch algorithm. However, a true mini-batch algorithm should have a running time sublinear in n, preferably only polylogarithmic. We show that the recursive expression can be *truncated*, achieving a fast update time of $\widetilde{O}(kb^2)$ while only incurring a small additive error compared to the untruncated version[2].

In Section 5 we go on to provide theoretical guarantees for our algorithm. This is somewhat tricky for mini-batch algorithms due to their stochastic nature, as they may not even converge to a local-minima. To overcome this hurdle, we take the approach of Schwartzman (2023) and answer the question: how long does it take truncated mini-batch kernel k-means to terminate with an *early stopping condition*. Specifically, we terminate the algorithm when the improvement on the batch drops below some user provided parameter, ϵ. Early stopping conditions are very common in practice (e.g., sklearn (Pedregosa et al., 2011)).

We show that applying the k-means++ initialization scheme (Arthur & Vassilvitskii, 2007) for our initial centers implies we achieve the same approximation ratio, $O(\log k)$ in expectation, as the full-batch algorithm.

While our general approach is similar to (Schwartzman, 2023), we must deal with the fact that $\mathcal{H}$ may have an *infinite* dimension. The guarantees of (Schwartzman, 2023) depend on the dimension of the space in which k-means is executed, which is unacceptable in our case. We overcome this by parameterizing our results by a new parameter $\gamma = \max_{x\in X}\|\phi(x)\|$. We note that for normalized kernels, such as the popular Gaussian and Laplacian kernels, it holds that $\gamma = 1$. We also observe that it is often the case that $\gamma \ll 1$ for various other kernels used in practice (Appendix C). We show

[2] Where $\widetilde{O}$ hides factors that are polylogarithmic in $n, 1/\epsilon, \gamma$.

that if the batch size is $\Omega(\max\left\{\gamma^4, \gamma^2\right\} k\epsilon^{-2}\log^2(\gamma n/\epsilon))$ then w.h.p. our algorithm terminates in $O(\gamma^2/\epsilon)$ iterations. Our theoretical results are summarised in Theorem 1 (where Algorithm 2 is explained in Section 4 and the pseudo-code appears in Appendix A).

Theorem 1. *The following holds for Algorithm 2: (1) Each iteration takes $O(kb^2\log^2(\gamma/\epsilon))$ time, (2) If $b = \Omega(\max\left\{\gamma^4, \gamma^2\right\} k\epsilon^{-2}\log^2(\gamma n/\epsilon))$ then it terminates in $O(\gamma^2/\epsilon)$ iterations w.h.p, (3) When initialized with k-means++ it achieve a $O(\log k)$ approximation ratio in expectation.*

Our result improves upon (Schwartzman, 2023) significantly when a normalized kernel is used since Theorem 1 doesn't depend on the input dimension. Our algorithm copes better with non linearly separable data and requires a smaller batch size ($\widetilde{\Omega}(1/\epsilon^2)$ vs $\widetilde{\Omega}((d/\epsilon)^2)$)) [3] for normalized kernels. This is particularly apparent with high dimensional datasets such as MNIST (LeCun, 1998) where the dimension squared is already nearly ten times the number of datapoints.

The learning rate we use, suggested in (Schwartzman, 2023), differs from the standard learning rate of sklearn in that it does not go to 0 over time. Unfortunately, this new learning rate is non-standard and (Schwartzman, 2023) did not present experiments comparing their learning rate to that of sklearn.

In Section 6 we extensively evaluate our results experimentally both with the learning rate of (Schwartzman, 2023) and that of sklearn. We also fill the experimental gap left in (Schwartzman, 2023) by evaluating (non-kernel) mini-batch k-means with their new learning rate compared to that of sklearn. To allow a fair empirical comparison, we run each algorithm for a fixed number of iterations without stopping conditions. Our results are as follows: 1) Truncated mini-batch kernel k-means is significantly faster than full-batch kernel k-means, while achieving solutions of similar quality, which are superior to the non-kernel version, 2) The learning rate of (Schwartzman, 2023) results in solutions with better quality both for truncated mini-batch kernel k-means and (non-kernel) mini-batch k-means.

2 Related work

Until recently, mini-batch k-means was only considered with a learning rate going to 0 over time. This was true both in theory (Tang & Monteleoni, 2017; Sculley, 2010) and practice (Pedregosa et al., 2011). Recently, (Schwartzman, 2023) proposed a new learning which does not go to 0 over time, and showed that if the batch is of size $\widetilde{\Omega}(k(d/\epsilon)^2)$[4], mini-batch k-means must terminate within $O(d/\epsilon)$ iterations with high probability, where d is the dimension of the input, and ϵ is a threshold parameter for termination.

A popular approach to deal with the slow running time of kernel k-means is constructing a *coreset* of the data. A coreset for kernel k-means is a weighted subset of X with the guarantee that the solution quality on the coreset is close to that on the entire dataset up to a $(1+\epsilon)$ multiplicative factor. There has been a long line of work on coresets for k-means an kernel k-means (Schmidt, 2014; Feldman et al., 2020; Barger & Feldman, 2020), and the current state-of-the-art for kernel k-means is due to (Jiang et al., 2021). They present a coreset algorithm with a nearly linear (in n and k) construction time which outputs a coreset of size $poly(k\epsilon^{-1})$.

In (Chitta et al., 2011) the authors only compute the kernel matrix for uniformly sampled set of m points from X. Then they optimize a variant of kernel k-means where the centers are constrained to be linear combinations of the sampled points. The authors do no provide worst case guarantees for the running time or approximation of their algorithm.

Another approach to speed up kernel k-means is by computing an approximation for the kernel matrix. This can be done by computing a low dimensional approximation for ϕ (without computing ϕ explicitly)(Rahimi & Recht, 2007; Chitta et al., 2012; Chen & Phillips, 2017), or by computing a low rank approximation for the kernel matrix (Musco & Musco, 2017; Wang et al., 2019).

[3] In (Schwartzman, 2023) the tilde notation hides factors logarithmic in d instead of γ.

[4] The original paper of (Schwartzman, 2023) states the batch size as $\widetilde{\Omega}((d/\epsilon)^2)$, however there is a mistake in the calculations which requires an additional k factor. We explain the issue in the proof of Lemma 15.

Kernel sparsification techniques construct sparse approximations of the full kernel matrix in subquadratic time. For smooth kernel functions such as the polynomial kernel, (Quanrud) presents an algorithm for constructing a $(1+\epsilon)$-spectral sparsifier for the full kernel matrix with a nearly linear number of non-zero entries in nearly linear time. For the gaussian kernel, (Macgregor & Sun, 2024) show how to construct a weaker, cluster preserving sparsifier using a nearly linear number of kernel density estimation queries.

We note that our results are *complementary* to coresets, dimensionality reduction, and kernel sparsification, in the sense that we can compose our method with these techniques.

3 Preliminaries

Throughout this paper we work with ordered tuples rather than sets, denoted as $Y = (y_i)_{i\in[\ell]}$, where $[\ell] = \{1, \dots, \ell\}$. To reference the i-th element we either write y_i or $Y[i]$. It will be useful to use set notations for tuples such as $x \in Y \iff \exists i \in [\ell], x = y_i$ and $Y \subseteq Z \iff \forall i \in [\ell], y_i \in Z$. When summing we often write $\sum_{x\in Y} g(x)$ which is equivalent to $\sum_{i=1}^{\ell} g(Y[i])$.

We borrow the following notation from (Kanungo et al., 2004) and generalize it to Hilbert spaces. For every $x, y \in \mathcal{H}$ let $\Delta(x, y) = \|x-y\|^2$. We slightly abuse notation and and also write $\Delta(x, y) = \|\phi(x) - \phi(y)\|^2$ when $x, y \in X$ and $\Delta(x, y) = \|\phi(x) - y\|^2$ when $x \in X, y \in \mathcal{H}$ (similarly when $x \in \mathcal{H}, y \in X$). For every finite tuple $S \subseteq X$ and a vector $x \in \mathcal{H}$ let $\Delta(S, x) = \sum_{y\in S} \Delta(y, x)$. Let us denote $\gamma = \max_{x\in X} \|\phi(x)\|$. Let us define for any finite tuple $S \subseteq X$ the center of mass of the tuple as $cm(S) = \frac{1}{|S|}\sum_{x\in S} \phi(x)$.

We now state the kernel k-means problem using the above notation.

Kernel k-means We are given an input $X = (x_i)_{i=1}^n$ and a parameter k. Our goal is to (implicitly) find a tuple $\mathcal{C} \subseteq \mathcal{H}$ of k centers such that the following goal function is minimized: $\frac{1}{n}\sum_{x\in X} \min_{C\in\mathcal{C}} \Delta(x, C)$.

Let us define for every $x \in X$ the function $f_x : \mathcal{H}^k \to \mathbb{R}$ where $f_x(\mathcal{C}) = \min_{C\in\mathcal{C}} \Delta(x, C)$. We can treat $\mathcal{H}^k$ as the set of k-tuples of vectors in $\mathcal{H}$. We also define the following function for every tuple $A = (a_i)_{i=1}^{\ell} \subseteq X$: $f_A(\mathcal{C}) = \frac{1}{\ell}\sum_{i=1}^{\ell} f_{a_i}(\mathcal{C})$. Note that f_X is our original goal function.

We make extensive use of the notion of *convex combination*:

Definition 2. *We say that $y \in \mathcal{H}$ is a* convex combination of X *if $y = \sum_{x\in X} p_x\phi(x)$, such that $\forall x \in X, p_x \geq 0$ and $\sum_{x\in X} p_x = 1$.*

4 Our Algorithm

We start by presenting a slower algorithm that will set the stage for our truncated mini-batch algorithm and will be useful during the analysis. We present our pseudo-code in Algorithm 1. It requires an initial set of cluster centers such that every center is a convex combination of X. This guarantees that all subsequent centers are also a convex combination of X. Note that if we initialize the centers using the kernel version of k-means++, this is indeed the case.

Algorithm 1 proceeds by repeatedly sampling a batch of size b (the batch size is a parameter). For the i-th batch the algorithm (implicitly) updates the centers using the learning rate α_j^i for center j. Note that the learning rate may take on different values for different centers, and may change between iterations. Finally, the algorithm terminates when the progress on the batch is below ϵ, a user provided parameter. While our termination guarantees (Section 5) require a specific learning rate, it does not affect the running time of a single iteration, and we leave it as a parameter for now.

Recursive distance update rule While for (non kernel) k-means the center updates and assignment of points to clusters is straightforward, this is tricky for kernel k-means and even harder for mini-batch kernel k-means. Specifically, how do we overcome the challenge that we do not maintain the centers explicitly?

To assign points to centers in the $(i+1)$-th iteration, it is sufficient to know $\|\phi(x) - \mathcal{C}_{i+1}^j\|^2$ for every j. If we can keep track of this quantity through the execution of the algorithm, we are done.

```
Algorithm 1: Mini-batch kernel $k$-means with early stopping
Input:
• Dataset $X = (x_i)_{i=1}^n$, batch size $b$, early stopping parameter $\epsilon$
• Initial centers $(\mathcal{C}_1^j)_{j=1}^k$ where $\mathcal{C}_1^j$ is a convex combination of $X$ for all $j \in [k]$
for $i = 1$ to $\infty$ do
    Sample $b$ elements, $B_i = (y_1, \ldots, y_b)$, uniformly at random from $X$ (with repetitions)
    for $j = 1$ to $k$ do
        $B_i^j = \{x \in B_i \mid \arg\min_{\ell \in [k]} \Delta(x, \mathcal{C}_i^\ell) = j\}$
        $\alpha_i^j = \sqrt{\left|B_i^j\right| / b}$ is the learning rate for the $j$-th cluster for iteration $i$
        $\mathcal{C}_{i+1}^j = (1 - \alpha_i^j)\mathcal{C}_i^j + \alpha_i^j cm(B_i^j)$
    if $f_{B_i}(\mathcal{C}_{i+1}) - f_{B_i}(\mathcal{C}_i) < \epsilon$ then Return $\mathcal{C}_{i+1}$
```

Let us derive a *recursive* expression for the distances

$$\|\phi(x) - \mathcal{C}_{i+1}^j\|^2 = \langle \phi(x), \phi(x) \rangle - 2\langle \phi(x), \mathcal{C}_{i+1}^j \rangle + \langle \mathcal{C}_{i+1}^j, \mathcal{C}_{i+1}^j \rangle.$$

We first expand $\langle \phi(x), \mathcal{C}_{i+1}^j \rangle$,

$$\langle \phi(x), \mathcal{C}_{i+1}^j \rangle = \langle \phi(x), (1 - \alpha_i^j)\mathcal{C}_i^j + \alpha_i^j cm(B_i^j) \rangle = (1 - \alpha_i^j)\langle \phi(x), \mathcal{C}_i^j \rangle + \alpha_i^j \langle \phi(x), cm(B_i^j) \rangle.$$

Then we expand $\langle \mathcal{C}_{i+1}^j, \mathcal{C}_{i+1}^j \rangle$,

$$\begin{aligned}\langle \mathcal{C}_{i+1}^j, \mathcal{C}_{i+1}^j \rangle &= \langle (1 - \alpha_i^j)\mathcal{C}_i^j + \alpha_i^j cm(B_i^j), (1 - \alpha_i^j)\mathcal{C}_i^j + \alpha_i^j cm(B_i^j) \rangle \\ &= (1 - \alpha_i^j)^2 \langle \mathcal{C}_i^j, \mathcal{C}_i^j \rangle + 2\alpha_i^j(1 - \alpha_i^j)\langle \mathcal{C}_i^j, cm(B_i^j) \rangle + (\alpha_i^j)^2 \langle cm(B_i^j), cm(B_i^j) \rangle.\end{aligned}$$

The above is all we need to compute the distances. Furthermore, it is possible to use dynamic programming to update the center for every iteration in $O(n(b + k))$ time and $O(nk)$ space (Appendix A). This is a considerable speedup compared to the best known quadratic update time. Next, we go a step further and show that it is possible to get an update time with only polylogarithmic dependence on n.

4.1 TRUNCATING THE CENTERS

The issue with the above approach is that each center is written as linear combination of potentially all points in X. We now present a simple way to overcome this issue. We maintain $\mathcal{C}_{i+1}^j$ as an explicit sparse linear combination of X. Let us expand the recursive expression of $\mathcal{C}_{i+1}^j$ for t terms, assuming $t > i$:

$$\mathcal{C}_{i+1}^j = (1 - \alpha_i^j)\mathcal{C}_i^j + \alpha_i^j cm(B_i^j) = \mathcal{C}_{i-t}^j \Pi_{\ell=0}^t (1 - \alpha_{i-\ell}^j) + \sum_{\ell=0}^t \alpha_{i-\ell}^j cm(B_{i-\ell}^j) \Pi_{z=i-\ell+1}^i (1 - \alpha_z^j).$$

The idea behind our truncation technique is that when t is sufficiently large $\mathcal{C}_{i-t}^j \Pi_{\ell=0}^t (1 - \alpha_{i-\ell}^j)$ becomes very small and can be discarded. The rate by which this term decays depends on the learning rates, which in turn depend on the number of elements assigned to the cluster in each of the previous iterations.

Let us denote $b_i^j = \left|B_i^j\right|$. We would like to trim the recursive expression such that every cluster center is represented using about τ points, where τ is a parameter. We define Q_i^j to be the set of indices from i to $i - t$, where t is the smallest integer such that $\sum_{\ell \in Q_i^j} b_i^j \geq \tau$ holds. If no such integer exists then $Q_i^j = \{i, i - 1, \ldots, 1\}$. It is the case that $\sum_{\ell \in Q_i^j} b_i^j \leq \tau + b$.

Next we define the *truncated centers*, for which the contributions of older points to the centers are forgotten after about τ points have been assigned to the center:

$$\widehat{\mathcal{C}}^j_{i+1} = \begin{cases} \sum_{\ell\in Q_i^j} \alpha_\ell^j cm(B_\ell^j) \prod_{\ell\in Q_i^j\setminus\{i\}}(1-\alpha_\ell^j), & \min Q_i^j > 1 \\ \mathcal{C}^j_{i+1} & \text{otherwise.} \end{cases} \tag{1}$$

From the above definition it is always the case that either $\mathcal{C}^j_{i+1} = \widehat{\mathcal{C}}^j_{i+1}$ or $\sum_{\ell\in Q_i^j} b_i^j \geq \tau$. The following lemma shows that when τ is sufficiently large $\|\widehat{\mathcal{C}}^j_{i+1} - \mathcal{C}^j_{i+1}\|$ is small. Intuitively, this implies that the truncated algorithm should achieve results similar to the untruncated version (we formalize this intuition in Section 5).

Lemma 3. *Setting $\tau = \lceil b\ln^2(28\gamma/\epsilon)\rceil$ it holds that $\forall i \in \mathbb{N}, j\in[k], \|\widehat{\mathcal{C}}^j_{i+1} - \mathcal{C}^j_{i+1}\| \leq \epsilon/28$.*

Proof. We assume that $\sum_{\ell\in Q_i^j} b_i^j \geq \tau$, as otherwise the claim trivially holds.

$$\|\widehat{\mathcal{C}}^j_{i+1} - \mathcal{C}^j_{i+1}\| = \|\mathcal{C}^j_{\min\{Q_i^j\}}\Pi_{\ell\in Q_i^j}(1-\alpha_\ell^j)\| \leq \|\mathcal{C}^j_{\min\{Q_i^j\}}\|e^{-\sum_{\ell\in Q_i^j}\alpha_\ell^j}$$

It holds that $\sum_{\ell\in Q_i^j}\alpha_\ell^j = \sum_{\ell\in Q_i^j}\sqrt{b_\ell^j/b} \geq \sqrt{\sum_{\ell\in Q_i^j} b_\ell^j/b} \geq \sqrt{\tau/b} \geq \ln(28\gamma/\epsilon)$. Plugging this back into the exponent, we get that: $\|\mathcal{C}^j_{\min\{Q_i^j\}}\|e^{-\sum_{\ell\in Q_i^j}\alpha_\ell^j} \leq \gamma e^{\ln(\epsilon/28\gamma)} \leq \epsilon/28$. □

Algorithm implmentation and runtime To implement this, we simply need to swap $\mathcal{C}_i^j$ in Algorithm 1 with $\widehat{\mathcal{C}}_i^j$ (Lines 7 and 8). As before, the main bottleneck of each iteration, is assigning points in the batch to their closest center. Once this is done, updating the truncated centers is straightforward by simply adjusting the coefficients in equation 1, removing the last element from the sum and adding a new element to the sum[5]. If $\min Q_i^j$ is 1, then we also need to add $\mathcal{C}_1^j\Pi_{\ell\in Q_i^j}(1-\alpha_\ell^j)$ which guarantees that $\widehat{\mathcal{C}}_i^j = \mathcal{C}_i^j$. The pseudo code is provided in Algorithm 2, and is deferred to Appendix A, as it is almost identical to Algorithm 1.

As before, let us consider assigning all points in the $(i+1)$ iteration to their closest centers. Unlike the previous approach, when computing distances between points in B_{i+1} and $\widehat{\mathcal{C}}_{i+1}$ we can do this directly (without recursion) and it is now sufficient to consider a much smaller set of inner products.

As before, the terms we are interested in computing are: $\langle\phi(x), \widehat{\mathcal{C}}^j_{i+1}\rangle$ and $\langle\widehat{\mathcal{C}}^j_{i+1}, \widehat{\mathcal{C}}^j_{i+1}\rangle$. However, there are several differences to the previous approach. We no longer need $\langle\phi(x), \widehat{\mathcal{C}}^j_{i+1}\rangle$ for all $x\in X$, but only for $x \in B_{i+1}$. Furthermore, $\widehat{\mathcal{C}}^j_{i+1}$ can be simply written as a weighted sum of at most $\sum_{\ell\in Q_i^j} b_\ell^j \leq \tau + b$ terms. Summing over all element in B_{i+1} and k centers we get $O(kb(b+\tau))$ time to compute $\langle\phi(x), \widehat{\mathcal{C}}^j_{i+1}\rangle$. For $\langle\widehat{\mathcal{C}}^j_{i+1}, \widehat{\mathcal{C}}^j_{i+1}\rangle$ using the bound on the number of terms we directly get $O(k(\tau+b)^2)$ time. We conclude that every iteration of Algorithm 2 requires $O(k(\tau+b)^2) = \widetilde{O}(kb^2)$ time. The additional space required is $O(k\tau) = \widetilde{O}(kb)$.

5 Termination guarantee

In this section we prove the second claim of Theorem 1. For most of the section we analyze Algorithm 1, and towards the end we use the fact that the centers of the two algorithms are close throughout the execution to conclude our proof.

Section preliminaries We introduce the following definitions and lemmas to aid our proof of the second claim of Theorem 1.

Lemma 4. *For every y which is a convex combination of X it holds that $\|y\| \leq \gamma$.*

[5] In our code we use an efficient sliding window implementation to store and update coefficients.

Proof. The proof follows by a simple application of the triangle inequality:

$$\|y\| = \| \sum_{x\in X} p_x \phi(x)\| \le \sum_{x\in X} \|p_x\phi(x)\| = \sum_{x\in X} p_x \|\phi(x)\| \le \sum_{x\in X} p_x \gamma = \gamma. \square$$

Lemma 5. *For any tuple of k centers $\mathcal{C} \subset \mathcal{H}^d$ which are a convex combination of points in X, it holds that $\forall A \subseteq X, f_A(\mathcal{C}) \le 4\gamma^2$.*

Proof. It is sufficient to upper bound f_x. Combining that fact that every $C \in \mathcal{C}$ is a convex combination of X with the triangle inequality, we have that

$$\forall x \in X, f_x(\mathcal{C}) \le \max_{C\in\mathcal{C}} \Delta(x, C) = \Delta(x, \sum_{y\in X} p_y\phi(y))$$
$$= \|\phi(x) - \sum_{y\in X} p_y\phi(y)\|^2 \le (\|\phi(x)\| + \| \sum_{y\in X} p_y\phi(y)\|)^2 \le 4\gamma^2. \qquad \square$$

We state the following simplified version of an Azuma bound for Hilbert space valued martingales from (Naor, 2012), followed by a standard Hoeffding bound.

Theorem 6 ((Naor, 2012))**.** *Let $\mathcal{H}$ be a Hilbert space and let $Y_0, ..., Y_m$ be a $\mathcal{H}$-valued martingale, such that $\forall 1 \le i \le m, \|Y_i - Y_{i-1}\| \le a_i$. It holds that $Pr[\|Y_m - Y_0\| \ge \delta] \le e^{\Theta\left(\frac{\delta^2}{\sum_{i=1}^m a_i^2}\right)}$.*

Theorem 7 ((Hoeffding, 1963))**.** *Let $Y_1, ..., Y_m$ be independent random variables such that $\forall 1 \le i \le m, E[Y_i] = \mu$ and $Y_i \in [a_{min}, a_{max}]$. Then $Pr\left(\left|\frac{1}{m}\sum_{i=1}^m Y_k - \mu\right| \ge \delta\right) \le 2e^{-2m\delta^2/(a_{max}-a_{min})^2}$.*

The following lemma provides concentration guarantees when sampling a *batch*.

Lemma 8. *Let B be a tuple of b elements chosen uniformly at random from X with repetitions. For any fixed tuple of k centers, $\mathcal{C} \subseteq \mathcal{H}$ which are a convex combination of X, it holds that: $Pr[|f_B(\mathcal{C}) - f_X(\mathcal{C})| \ge \delta] \le 2e^{-b\delta^2/8\gamma^4}$.*

Proof. Let us write $B = (y_1, \ldots, y_b)$, where y_i is a random element selected uniformly at random from X with repetitions. For every such y_i define the random variable $Z_i = f_{y_i}(\mathcal{C})$. These new random variables are IID for any fixed $\mathcal{C}$. It also holds that $\forall i \in [b], E[Z_i] = \frac{1}{n}\sum_{x\in X} f_x(\mathcal{C}) = f_X(\mathcal{C})$ and that $f_B(\mathcal{C}) = \frac{1}{b}\sum_{x\in B} f_x(\mathcal{C}) = \frac{1}{b}\sum_{i=1}^b Z_i$.

Applying the Hoeffding bound (Theorem 7) with parameters $m = b, \mu = f_X(\mathcal{C}), a_{max} - a_{min} \le 4\gamma^2$ (due to Lemma 5) we get that: $Pr[|f_B(\mathcal{C}) - f_X(\mathcal{C})| \ge \delta] \le 2e^{-b\delta^2/8\gamma^4}$. $\square$

For any tuple $S \subseteq X$ and some tuple of cluster centers $\mathcal{C} = (\mathcal{C}^\ell)_{\ell\in[k]} \subset \mathcal{H}$, $\mathcal{C}$ implies a *partition* $(S^\ell)_{\ell\in[k]}$ of the points in S. Specifically, every S^ℓ contains the points in S closest to $\mathcal{C}^\ell$ (in $\mathcal{H}$) and every point in S belongs to a single $\mathcal{C}^\ell$ (ties are broken arbitrarily). We state the following useful observation:

Observation 9. *Fix some $A \subseteq X$. Let $\mathcal{C}$ be a tuple of k centers, $S = (S^\ell)_{\ell\in[k]}$ be the partition of A induced by $\mathcal{C}$ and $\overline{S} = (\overline{S}^\ell)_{\ell\in[k]}$ be any other partition of A. It holds that $\sum_{j=1}^k \Delta(S^j, \mathcal{C}^j) \le \sum_{j=1}^k \Delta(\overline{S}^j, \mathcal{C}^j)$.*

Recall that $\mathcal{C}_i^j$ is the j-th center in the beginning of the i-th iteration of Algorithm 1 and $(B_i^\ell)_{\ell\in[k]}$ is the partition of B_i induced by $\mathcal{C}_i$. Let $(X_i^\ell)_{\ell\in[k]}$ be the partition of X induced by $\mathcal{C}_i$.

We now have the tools to analyze Algorithm 1 with the learning rate of (Schwartzman, 2023). Specifically, we assume that the algorithm executes for at least t iterations, the learning rate is $\alpha_i^j = \sqrt{b_i^j/b}$, where $b_i^j = \left|B_i^j\right|$, and the batch size is $b = \Omega(\max\left\{\gamma^4, \gamma^2\right\} k\epsilon^{-2}\log(nt))$. We show that the algorithm must terminate within $t = O(\gamma^2/\epsilon)$ steps w.h.p. Plugging t back into b, we get that a batch size of $b = \Omega(\max\left\{\gamma^4, \gamma^2\right\} k\epsilon^{-2}\log^2(\gamma n/\epsilon))$ is sufficient. We assume that ϵ is chosen such that $\gamma^2/\epsilon > 1/4$. Otherwise, the stopping condition immediately holds due to Lemma 5.

Proof outline We note that when sampling a batch it holds w.h.p that $f_{B_i}(\mathcal{C}_i)$ is close to $f_{X_i}(\mathcal{C}_i)$ (Lemma 8). This is due to the fact that B_i is sampled after $\mathcal{C}_i$ is fixed. If we could show that $f_{B_i}(\mathcal{C}_{i+1})$ is close $f_{X_i}(\mathcal{C}_{i+1})$ then combined with the fact that we make progress of at least ϵ on the batch we can conclude that we make progress of at least some constant fraction of ϵ on the entire dataset.

Unfortunately, as $\mathcal{C}_{i+1}$ depends on B_i, getting the above guarantee is tricky. To overcome this issue we define the auxiliary value $\overline{\mathcal{C}}_{i+1}^j = (1-\alpha_i^j)\mathcal{C}_i^j + \alpha_i^j cm(X_i^j)$. This is the j-th center at step $i+1$ if we were to use the entire dataset for the update, rather than just a batch. Note that this is only used in the analysis and not in the algorithm. Note that $\overline{\mathcal{C}}_{i+1}$ is *almost* independent of B_i. Every $\overline{\mathcal{C}}_{i+1}^j$ depends only on $\mathcal{C}_i^j, X_i^j$ and α_i^j. While $\mathcal{C}_i^j, X_i^j$ are independent of B_i, the learning α_i^j *is not*. Nevertheless, the number of possible values of $\left\{\alpha_i^j\right\}_{j\in k}$ is sufficiently small, and we can overcome this issue by showing concentration for every possible learning rate configuration followed by a union bound. This allows us to use $\overline{\mathcal{C}}_{i+1}$ instead of $\mathcal{C}_{i+1}$ in the above analysis outline. We show that for our choice of learning rate it holds that $\overline{\mathcal{C}}_{i+1}, \mathcal{C}_{i+1}$ are sufficiently close, which implies that $f_X(\mathcal{C}_{i+1}), f_X(\overline{\mathcal{C}}_{i+1})$ and $f_{B_i}(\mathcal{C}_{i+1}), f_{B_i}(\overline{\mathcal{C}}_{i+1})$ are also sufficiently close. That is, $\overline{\mathcal{C}}_{i+1}$ acts as a proxy for $\mathcal{C}_{i+1}$. Combining everything together we get our desired result for Algorithm 1.

We start with the following useful observation, which will allow us to use Lemma 4 to bound the norm of the centers by γ throughout the execution of the algorithm.

Observation 10. *If $\forall j \in [k], \mathcal{C}_1^j$ is a convex combination of X then $\forall i > 1, j \in [k], \mathcal{C}_i^j, \overline{\mathcal{C}}_i^j$ are also a convex combinations of X.*

Let us state the following useful lemma from (Kanungo et al., 2004). Although their proof is for Euclidean spaces, it goes through for Hilbert spaces. We provide the proof in the appendix for completeness.

Lemma 11 ((Kanungo et al., 2004))**.** *For any set $S \subseteq X$ and any $C \in \mathcal{H}$ it holds that $\Delta(S, C) = \Delta(S, cm(S)) + |S|\,\Delta(C, cm(S))$.*

We use the above to state the following useful lemma (proof is deferred to Appendix B.

Lemma 12. *For any $S \subseteq X$ and $C, C' \in \mathcal{H}$ which are convex combinations of X, it holds that: $|\Delta(S, C') - \Delta(S, C)| \leq 4\gamma\,|S|\,\|C - C'\|$.*

We use the above to show that when centers are sufficiently close, their values are close for any f_A.

Lemma 13. *Fix some $A \subseteq X$ and let $(\mathcal{C}^j)_{j\in[k]}, (\overline{\mathcal{C}}^j)_{j\in[k]} \subset \mathcal{H}$ be arbitrary centers such that $\forall j \in [k], \|\mathcal{C}^j - \overline{\mathcal{C}}^j\| \leq \epsilon/28\gamma$. It holds that $\forall i \in [t], \left|f_A(\overline{\mathcal{C}}_{i+1}) - f_A(\mathcal{C}_{i+1})\right| \leq \epsilon/7$.*

Proof. Let $S = (S^\ell)_{\ell\in[k]}, \overline{S} = (\overline{S}^\ell)_{\ell\in[k]}$ be the partitions induced by $\mathcal{C}, \overline{\mathcal{C}}$ on A. Let us expand the expression

$$f_A(\overline{\mathcal{C}}) - f_A(\mathcal{C}) = \frac{1}{|A|}\sum_{j=1}^{k} \Delta(\overline{S}^j, \overline{\mathcal{C}}^j) - \Delta(S^j, \mathcal{C}^j) \leq \frac{1}{|A|}\sum_{j=1}^{k} \Delta(S^j, \overline{\mathcal{C}}^j) - \Delta(S^j, \mathcal{C}^j)$$

$$\leq \frac{1}{|A|}\sum_{j=1}^{k} 4\gamma\,\left|S^j\right|\,\|\overline{\mathcal{C}}^j - \mathcal{C}^j\| \leq \frac{1}{|A|}\sum_{j=1}^{k} \left|S^j\right|\epsilon/7 = \epsilon/7.$$

Where the first inequality is due to Observation 9, the second is due Lemma 12 and finally we use the assumption about the distances between centers together with the fact that $\sum_{j=1}^{k}\left|S^j\right| = |A|$. Using the same argument we also get that $f_A(\mathcal{C}) - f_A(\overline{\mathcal{C}}) \leq \epsilon/7$, which completes the proof. □

Now we show that due to our choice of learning rate, $\mathcal{C}_{i+1}^j$ and $\overline{\mathcal{C}}_{i+1}^j$ are sufficiently close.

Lemma 14. *It holds w.h.p that $\forall i \in [t], j \in [k], \|\mathcal{C}_{i+1}^j - \overline{\mathcal{C}}_{i+1}^j\| \leq \frac{\epsilon}{28\gamma}$.*

Proof. Note that $\mathcal{C}^j_{i+1} - \overline{\mathcal{C}}^j_{i+1} = \alpha^j_i(cm(B^j_i) - cm(X^j_i))$. Let us fix some iteration i and center j. To simplify notation, let us denote: $X' = X^j_i, B' = B^j_i, b' = b^j_i, \alpha' = \alpha^j_i$. Although b' is a random variable, in what follows we treat it as a fixed value (essentially conditioning on its value). As what follows holds for *all* values of b' it also holds without conditioning due to the law of total probabilities.

For the rest of the proof, we assume $b' > 0$ (if $b' = 0$ the claim holds trivially). Let us denote by $\{Y_\ell\}^{b'}_{\ell=1}$ the sampled points in B'. Note that a randomly sampled element from X is in B' if and only if it is in X'. As batch elements are sampled uniformly at random with repetitions from X, conditioning on the fact that an element is in B' means that it is distributed uniformly over X'. Note that $\forall \ell, E[\phi(Y_\ell)] = \frac{1}{|X'|}\sum_{x \in X'} \phi(x) = cm(X')$ and $E[cm(B')] = \frac{1}{b'}\sum^{b'}_{\ell=1} E[\phi(Y_\ell)] = cm(X')$.

Let us define the following martingale: $Z_r = \sum^r_{\ell=1}(\phi(Y_\ell) - E[\phi(Y_\ell)])$. Note that $Z_0 = 0$, and when $r > 0$, $Z_r = \sum^r_{\ell=1} \phi(Y_\ell) - r \cdot cm(X')$. It is easy to see that this is a martingale:

$$E[Z_r \mid Z_{r-1}] = E[\sum^{r}_{\ell=1} \phi(Y_\ell) - r \cdot cm(X') \mid Z_{r-1}] = Z_{r-1} + E[\phi(Y_r) - cm(X') \mid Z_{r-1}] = Z_{r-1}.$$

We bound the differences: $\|Z_r - Z_{r-1}\| = \|\phi(Y_r) - cm(X')\| \leq \|\phi(Y_r)\| + \|cm(X')\| \leq 2\gamma$.

Now we may use Azuma's inequality: $Pr[\|Z_{b'} - Z_0\| \geq \delta] \leq e^{-\Theta(\frac{\delta^2}{\gamma^2 b'})}$. Let us now divide both sides of the inequality by b' and set $\delta = \frac{b'\epsilon}{28\gamma\alpha'}$. We get

$$Pr[\|cm(B') - cm(X')\| \geq \frac{\epsilon}{28\gamma\alpha'}] = Pr[\|\frac{1}{b'}\sum^{b'}_{\ell=1} \phi(Y_\ell) - cm(X')\| \geq \frac{\epsilon}{28\gamma\alpha'}] \leq e^{-\Theta(\frac{b'\epsilon^2}{(\gamma\alpha')^2})}.$$

Using the fact that $\alpha' = \sqrt{b'/b}$ together with the fact that $b = \Omega(\max\{\gamma^4, \gamma^2\} k\epsilon^{-2} \log(nt))$ (for an appropriate constant) we get that the above is $O(1/ntk)$. Finally, taking a union bound over all t iterations and all k centers per iteration completes the proof. □

Let us state the following useful lemma.

Lemma 15. *It holds w.h.p that for every $i \in [t]$,*

$$f_X(\overline{\mathcal{C}}_{i+1}) - f_X(\mathcal{C}_{i+1}) \geq -\epsilon/7 \tag{2}$$

$$f_{B_i}(\mathcal{C}_{i+1}) - f_{B_i}(\overline{\mathcal{C}}_{i+1}) \geq -\epsilon/7 \tag{3}$$

$$f_X(\mathcal{C}_i) - f_{B_i}(\mathcal{C}_i) \geq -\epsilon/7 \tag{4}$$

$$f_{B_i}(\overline{\mathcal{C}}_{i+1}) - f_X(\overline{\mathcal{C}}_{i+1}) \geq -\epsilon/7 \tag{5}$$

Proof. The first two inequalities follow from Lemma 13. The third is due to Lemma 8 by setting $\delta = \epsilon/7, B = B_i$:

$$Pr[|f_{B_i}(\mathcal{C}_i) - f_X(\mathcal{C}_i)| \geq \delta] \leq 2e^{-b\delta^2/8\gamma^4} = e^{-\Theta(b\epsilon^2/\gamma^4)} = e^{-\Omega(\log(nt))} = O(1/nt).$$

Where the last inequality is due to the fact that $b = \Omega(\max\{\gamma^4, \gamma^2\} k\epsilon^{-2} \log(nt))$ (for an appropriate constant).

The last inequality is a bit more involved[6]. Let $\vec{\ell} \in \mathbb{N}^k$ be a vector whose entries sum to b. For every $\vec{\ell}$ we can define $\overline{\mathcal{C}}_{i+1}(\vec{\ell})$ such that $\overline{\mathcal{C}}^j_{i+1}(\vec{\ell}) = \mathcal{C}^j_i(1 - \sqrt{\ell_j/b}) + \sqrt{\ell_j/b} \cdot cm(X^j_i)$. For every choice of $\vec{\ell}$ it holds that $\overline{\mathcal{C}}_{i+1}(\vec{\ell})$ is independent of B_i and we can apply Lemma 8 for every possible $\overline{\mathcal{C}}_{i+1}(\vec{\ell})$ by setting $\delta = \epsilon/7, B = B_i$

$$Pr[\left|f_{B_i}(\overline{\mathcal{C}}_{i+1}(\vec{\ell})) - f_X(\overline{\mathcal{C}}_{i+1}(\vec{\ell}))\right| \geq \delta] \leq 2e^{-b\delta^2/8\gamma^4} = e^{-\Theta(b\epsilon^2/\gamma^4)} = e^{-\Omega(k\log(nt))} = O(1/(nt)^k).$$

Where the last inequality is due to the fact that $b = \Omega(\max\{\gamma^4, \gamma^2\} k\epsilon^{-2} \log(nt))$ (for an appropriate constant). Finally, we take a union bound over all possible vectors $\vec{\ell}$, a total of $\binom{b+k-1}{k-1} \leq (\frac{(b+k-1)\cdot e}{k-1})^{k-1} = O(n^{k-1})$. As $\overline{\mathcal{C}}_{i+1}$ corresponds to at least one $\overline{\mathcal{C}}_{i+1}(\vec{\ell})$ we are done.

[6] In (Schwartzman, 2023) this case is treated the same as the third inequality, which is incorrect. Using our approach the analysis can be fixed, with an additional multiplicative k factor in the batch size.

Finally, taking a union bound over all t iterations we get the desired result. □

Putting everything together We wish to lower bound $f_X(\mathcal{C}_i) - f_X(\mathcal{C}_{i+1})$. We write the following, where the $\pm$ notation means we add and subtract a term:

$$\begin{aligned}
&f_X(\mathcal{C}_i) - f_X(\mathcal{C}_{i+1}) = f_X(\mathcal{C}_i) \pm f_{B_i}(\mathcal{C}_i) - f_X(\mathcal{C}_{i+1}) \\
&\geq f_{B_i}(\mathcal{C}_i) - f_X(\mathcal{C}_{i+1}) - \epsilon/7 = f_{B_i}(\mathcal{C}_i) \pm f_{B_i}(\mathcal{C}_{i+1}) - f_X(\mathcal{C}_{i+1}) - \epsilon/7 \\
&\geq f_{B_i}(\mathcal{C}_{i+1}) - f_X(\mathcal{C}_{i+1}) + 6\epsilon/7 \\
&= f_{B_i}(\mathcal{C}_{i+1}) \pm f_{B_i}(\overline{\mathcal{C}}_{i+1}) \pm f_X(\overline{\mathcal{C}}_{i+1}) - f_X(\mathcal{C}_{i+1}) + 6\epsilon/7 \geq 3\epsilon/7.
\end{aligned}$$

Where the first inequality is due to inequality 4 in Lemma 15 ($f_X(\mathcal{C}_i) - f_{B_i}(\mathcal{C}_i) \geq -\epsilon/7$), the second is due to the stopping condition of the algorithm ($f_{B_i}(\mathcal{C}_i) - f_{B_i}(\mathcal{C}_{i+1}) > \epsilon$), and the last is due to the remaining inequalities in Lemma 15. The above holds w.h.p over all of the iterations of the algorithms. Using these guarantees for Algorithm 1 we can easily derive our main result for the truncated version.

Truncated termination Using Lemma 3 together with Lemma 13 and the fact that $f_X(\mathcal{C}_i) - f_X(\mathcal{C}_{i+1}) \geq 3\epsilon/7$ we get that: $f_X(\widehat{\mathcal{C}}_i) - f_X(\widehat{\mathcal{C}}_{i+1}) \geq f_X(\mathcal{C}_i) - f_X(\mathcal{C}_{i+1}) - 2\epsilon/7 \geq \epsilon/7$. We conclude that when $b = \Omega(\max\left\{\gamma^4, \gamma^2\right\} k\epsilon^{-2} \log^2(\gamma n/\epsilon))$, w.h.p. Algorithm 2 terminates within $t = O(\gamma^2/\epsilon)$ iterations. This completes the second claim of Theorem 1. The final claim of Theorem 1 is due to the following lemma (proof deferred to Appendix B).

Lemma 16. *The expected approximation ratio of the solution returned by Algorithm 2 is at least the approximation guarantee of the initial centers provided to the algorithm.*

6 Experiments

We evaluate our mini-batch algorithms on the following datasets:

MNIST: The MNIST dataset (LeCun, 1998) has 70,000 grayscale images of handwritten digits (0 to 9), each image being 28x28 pixels. When flattened, this gives 784 features. **PenDigits:** The PenDigits dataset (Alpaydin & Alimoglu, 1998) has 10992 instances, each represented by an 16-dimensional vector derived from 2D pen movements. The dataset has 10 labelled clusters, one for each digit. **Letters:** The Letters dataset (Slate, 1991) has 20,000 instances of letters from 'A' to 'Z', each represented by 16 features. The dataset has 26 labelled clusters, one for each letter. **HAR:** The HAR dataset (Anguita et al., 2013) has 10,299 instances collected from smartphone sensors, capturing human activities like walking, sitting, and standing. Each instance is described by 561 features. The dataset has 6 labelled clusters, corresponding to different types of physical activities.

We compare the following algorithms: full-batch kernel k-means, truncated mini-batch kernel k-means, and mini-batch k-means (both kernel and non-kernel) with learning rates from (Schwartzman, 2023) and sklearn. We evaluate our results with batch sizes: 2048, 1024, 512, 256 and $\tau = 50, 100, 200, 300$. We execute every algorithm for 200 iterations. For the results below, we apply the Gaussian kernel: $K(x, y) = e^{-\|x-y\|^2/\kappa}$, where the κ parameter is set using the heuristic of (Wang et al., 2019) followed by some manual tuning (exact values appear in the supplementary materials). We also run experiments with the heat kernel and knn kernels. We repeat every experiment 10 times and present the average Adjusted Rand Index (ARI) (Gates & Ahn, 2017; Rand, 1971) and Normalized Mutual Information (NMI) (Lancichinetti et al., 2009) scores for every dataset. All experiments were conducted using an AMD Ryzen 9 7950X CPU with 128GB of RAM and a Nvidia GeForce RTX 4090 GPU. We present partial results in Figure 1 and the full results in Appendix C. Error bars in the plot measure the standard deviation.

Discussion Throughout our results we consistently observe that the truncated version achieves performance on par with the non-truncated version with a running time which is often an order of magnitude faster. Surprisingly, this often holds for tiny values of τ (e.g., 50) far below the theoretical threshold (i.e., $\tau \ll b$). We note that the analysis of our truncated algorithm relies heavily on the learning rate of (Schwartzman, 2023), which does not go to 0 (unlike that of sklearn), this essentially exponentially decays the contribution of points to their centers over time, while the learning rate of sklearn does not.

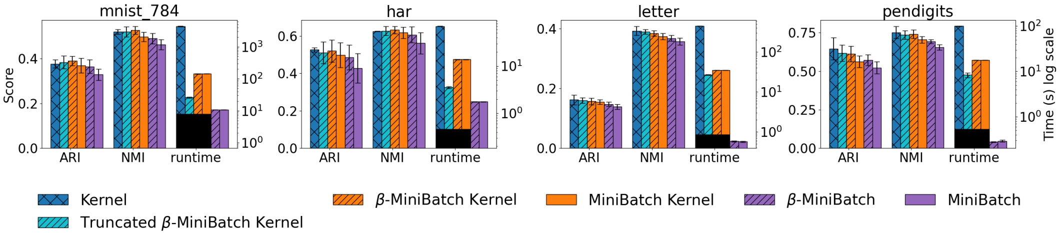

Figure 1: Our results for a batch size of size 1024 and $\tau = 200$ using the Gaussian kernel. We use the β prefix to denote the algorithm uses the learning rate of (Schwartzman, 2023). Black denotes the time required to compute the kernel.

References

E. Alpaydin and Fevzi. Alimoglu. Pen-Based Recognition of Handwritten Digits. UCI Machine Learning Repository, 1998. DOI: https://doi.org/10.24432/C5MG6K. License: CC BY 4.0 DEED, available at `https://creativecommons.org/licenses/by/4.0/`.

Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, Jorge Luis Reyes-Ortiz, et al. A public domain dataset for human activity recognition using smartphones. In *Esann*, volume 3, pp. 3, 2013. License: CC BY-NC-SA 4.0 DEED, available at `https://creativecommons.org/licenses/by-nc-sa/4.0/`.

David Arthur and Sergei Vassilvitskii. How slow is the *k*-means method? In *SCG*, pp. 144–153. ACM, 2006.

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

Artem Barger and Dan Feldman. Deterministic coresets for k-means of big sparse data. *Algorithms*, 13(4):92, 2020.

Di Chen and Jeff M Phillips. Relative error embeddings of the gaussian kernel distance. In *International Conference on Algorithmic Learning Theory*, 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 *2012 IEEE 12th International Conference on Data Mining*, pp. 161–170. IEEE, 2012.

Fan RK Chung. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997.

Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. Kernel k-means: spectral clustering and normalized cuts. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pp. 551–556, New York, NY, USA, 2004. Association for Computing Machinery. ISBN 1581138881. doi: 10.1145/1014052.1014118. URL `https://doi.org/10.1145/1014052.1014118`.

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

Alexander J Gates and Yong-Yeol Ahn. The impact of random models on clustering similarity. *Journal of Machine Learning Research*, 18(87):1–28, 2017.

Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

Shaofeng H.-C. Jiang, Robert Krauthgamer, Jianing Lou, and Yubo Zhang. Coresets for kernel clustering. *CoRR*, abs/2110.02898, 2021.

Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. A local search approximation algorithm for k-means clustering. *Comput. Geom.*, 28(2-3):89–112, 2004.

Andrea Lancichinetti, Santo Fortunato, and János Kertész. Detecting the overlapping and hierarchical community structure in complex networks. *New journal of physics*, 11(3):033015, 2009.

Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998. License: CC0 1.0 DEED CC0 1.0 Universal, available at `https://creativecommons.org/publicdomain/zero/1.0/`.

Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Trans. Inf. Theory*, 28(2):129–136, 1982.

Peter Macgregor and He Sun. Fast approximation of similarity graphs with kernel density estimation. *Advances in Neural Information Processing Systems*, 36, 2024.

Cameron Musco and Christopher Musco. Recursive sampling for the nystrom method. *Advances in neural information processing systems*, 30, 2017.

Assaf Naor. On the banach-space-valued azuma inequality and small-set isoperimetry of alon–roichman graphs. *Combinatorics, Probability and Computing*, 21(4):623–634, 2012.

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.

Kent Quanrud. *Spectral Sparsification of Metrics and Kernels*, pp. 1445–1464. doi: 10.1137/1.9781611976465.87. URL `https://epubs.siam.org/doi/abs/10.1137/1.9781611976465.87`.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20, 2007.

William M Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.

Melanie Schmidt. Coresets and streaming algorithms for the k-means problem and related clustering objectives. 2014.

Gregory Schwartzman. Mini-batch k-means terminates within O(d/ϵ) iterations. In *ICLR*, 2023.

D. Sculley. Web-scale k-means clustering. In *WWW*, pp. 1177–1178. ACM, 2010.

David Slate. Letter Recognition. UCI Machine Learning Repository, 1991. DOI: https://doi.org/10.24432/C5ZP40. License: CC BY 4.0 DEED, available at `https://creativecommons.org/licenses/by/4.0/`.

Cheng Tang and Claire Monteleoni. Convergence rate of stochastic k-means. In *AISTATS*, volume 54 of *Proceedings of Machine Learning Research*, pp. 1495–1503. PMLR, 2017.

Andrea Vattani. *k*-means requires exponentially many iterations even in the plane. *Discret. Comput. Geom.*, 45 (4):596–616, 2011.

Shusen Wang, Alex Gittens, and Michael W Mahoney. Scalable kernel k-means clustering with nystrom approximation: Relative-error bounds. *Journal of Machine Learning Research*, 20(12):1–49, 2019.

A Omitted proofs and Algorithms for Section 4

Runtime analysis of Algorithm 1 Assuming that $\langle \mathcal{C}_i^j, \mathcal{C}_i^j \rangle$ and $\langle \phi(x), \mathcal{C}_i^j \rangle$ are known for all $j \in [k]$ and for all $x \in X$, we can compute $\langle \mathcal{C}_{i+1}^j, \mathcal{C}_{i+1}^j \rangle$ and $\langle \phi(x), \mathcal{C}_{i+1}^j \rangle$ for all $j \in [k]$ and $x \in X$, which implies we can compute the distances from any point in the batch to all centers.

We now bound the running time of a single iteration of the outer loop in Algorithm 1. Let us denote $b_i^j = \left|B_i^j\right|$ and recall that $cm(B_i^j) = \frac{1}{b_i^j}\sum_{y \in B_i^j} \phi(y)$. Therefore, computing $\langle \phi(x), cm(B_i^j) \rangle = \frac{1}{b_i^j}\sum_{y \in B_i^j} \langle \phi(x), \phi(y) \rangle$ requires $O(b_i^j)$ time. Similarly, computing $\langle cm(B_i^j), cm(B_i^j) \rangle$ requires $O((b_i^j)^2)$ time. Let us now bound the time it requires to compute $\langle \phi(x), \mathcal{C}_{i+1}^j \rangle$ and $\langle \mathcal{C}_{i+1}^j, \mathcal{C}_{i+1}^j \rangle$.

Assuming we know $\langle \phi(x), \mathcal{C}_i^j \rangle$ and $\langle \mathcal{C}_i^j, \mathcal{C}_i^j \rangle$, updating $\langle \phi(x), \mathcal{C}_{i+1}^j \rangle$ for all $x \in X, j \in [k]$ requires $O(n(b+k))$ time. Specifically, the $\langle \phi(x), \mathcal{C}_i^j \rangle$ term is already known from the previous iteration and we need to compute $\alpha_i^j \langle \phi(x), cm(B_i^j) \rangle$ for every $x \in X, j \in [k]$ which requires $n \sum_{j \in [k]} b_i^j = nb$ time. Finally, updating $\langle \phi(x), \mathcal{C}_{i+1}^j \rangle$ for all $x \in X, j \in [k]$ requires $O(nk)$ time.

Updating $\langle \mathcal{C}^j_{i+1}, \mathcal{C}^j_{i+1}\rangle$ requires $O(b^2 + kb)$ time. Specifically, $\langle \mathcal{C}^j_i, \mathcal{C}^j_i\rangle$ is known from the previous iteration and computing $\langle cm(B^j_i), cm(B^j_i)\rangle$ for all $j \in [k]$ requires $O(\sum_{j\in[k]}(b^j_i)^2) = O(b^2)$ time. Computing $\langle \mathcal{C}^j_i, cm(B^j_i)\rangle$ for all $j \in [k]$ requires time $O(b)$ using $\langle \phi(x), \mathcal{C}^j_i\rangle$ from the previous iteration. Therefore, the total running time of the update step (assigning points to new centers) is $O(n(b + k))$. To perform the update at the $(i + 1)$-th step we only need $\langle \phi(x), \mathcal{C}^j_i\rangle$, $\langle \mathcal{C}^j_i, \mathcal{C}^j_i\rangle$, which results in a space complexity of $O(nk)$. This completes the first claim of Theorem 1.

Truncated mini-batch algorithm

Algorithm 2: Truncated Mini-batch kernel k-means with early stopping

```
for $i = 1$ to $\infty$ do
    Sample $b$ elements, $B_i = (y_1, \ldots, y_b)$, uniformly at random from $X$ (with repetitions)
    for $j = 1$ to $k$ do
        $B^j_i = \left\{x \in B_i \mid \arg\min_{\ell\in[k]} \Delta(x, \widehat{\mathcal{C}}^\ell_i) = j\right\}$
        $\alpha^j_i$ is the learning rate for the $j$-th cluster for iteration $i$
        $\widehat{\mathcal{C}}^j_{i+1} = \sum_{\ell\in Q^j_i} \alpha^j_\ell cm(B^j_\ell) \prod_{\ell\in Q^j_i}(1 - \alpha^j_\ell)$
        if $Q^j_i = 1$ then $\widehat{\mathcal{C}}^j_{i+1} = \widehat{\mathcal{C}}^j_{i+1} + \mathcal{C}^j_1 \Pi_{\ell\in Q^j_i\setminus\{i\}}(1 - \alpha^j_\ell)$
    if $f_{B_i}(\widehat{\mathcal{C}}_{i+1}) - f_{B_i}(\widehat{\mathcal{C}}_i) < \epsilon$ then Return $\widehat{\mathcal{C}}_{i+1}$
```

B Omitted proofs and Algorithms for Section 5

Proof of Lemma 11

Proof.

$$\begin{aligned}
\Delta(S, C) &= \sum_{x\in S} \Delta(x, C) = \sum_{x\in S} \langle x - C, x - C\rangle \\
&= \sum_{x\in S} \langle (x - cm(S)) + (cm(S) - C), (x - cm(S)) + (cm(S) - C)\rangle \\
&= \sum_{x\in S} \Delta(x, cm(S)) + \Delta(C, cm(S)) + 2\langle x - cm(S), cm(S) - C\rangle \\
&= \Delta(S, cm(S)) + |S|\,\Delta(C, cm(S)) + \sum_{x\in S} 2\langle x - cm(S), cm(S) - C\rangle \\
&= \Delta(S, cm(S)) + |S|\,\Delta(C, cm(S)),
\end{aligned}$$

where the last step is due to the fact that

$$\begin{aligned}
\sum_{x\in S} \langle x - cm(S), cm(S) - C\rangle &= \langle \sum_{x\in S} x - |S|\, cm(S), cm(S) - C\rangle \\
&= \langle \sum_{x\in S} x - \frac{|S|}{|S|} \sum_{x\in S} x, cm(S) - C\rangle = 0.
\end{aligned}$$

□

Proof of Lemma 12

Proof. Using Lemma 11 we get that $\Delta(S, C) = \Delta(S, cm(S)) + |S|\,\Delta(cm(S), C)$ and that $\Delta(S, C') = \Delta(S, cm(S)) + |S|\,\Delta(cm(S), C')$. Thus, it holds that $|\Delta(S, C') - \Delta(S, C)| = |S| \cdot$

$|\Delta(cm(S), C') - \Delta(cm(S), C)|$. Let us write

$$\begin{aligned}
&\big|\Delta(cm(S), C') - \Delta(cm(S), C)\big| \\
&= \big|\langle cm(S) - C', cm(S) - C'\rangle - \langle cm(S) - C, cm(S) - C\rangle\big| \\
&= \big|-2\langle cm(S), C'\rangle + \langle C', C'\rangle + 2\langle cm(S), C\rangle - \langle C, C\rangle\big| \\
&= \big|2\langle cm(S), C - C'\rangle + \langle C' - C, C' + C\rangle\big| \\
&= \big|\langle C - C', 2cm(S) - (C' + C)\rangle\big| \\
&\leq \|C - C'\|\|2cm(S) - (C' + C)\| \leq 4\gamma\|C - C'\|.
\end{aligned}$$

Where in the last transition we used the Cauchy-Schwartz inequality, the triangle inequality, and the fact that $C, C', cm(S)$ are convex combinations of X and therefore their norm is bounded by γ. □

Proof of Lemma 16

Proof. Let $p = 1 - O(\epsilon/n\gamma^2) = 1 - O(1/n)$ be the success probability of a single iteration. By "success" we mean that all inequalities in Lemma 15 hold. The value of p is due to the fact that we take $t = O(\gamma^2/\epsilon)$ and that $\gamma^2/\epsilon \geq 1/4$.

With probability at least p, it holds that $f_X(\mathcal{C}_{i+1}) \leq f_X(\mathcal{C}_i) - 2\epsilon/7$. On the other hand, f_X is upper bounded by $4\gamma^2$. Let us denote $Z = f_X(\mathcal{C}_i) - f_X(\mathcal{C}_{i+1})$ the change in the goal function after the i-th iteration. Consider the following:

$$E[Z] = E[Z \mid Z \geq \epsilon/7]Pr[Z \geq \epsilon/7] + E[Z \mid Z < \epsilon/7]Pr[Z < \epsilon/7]$$

We show that $E[Z] = E[f_X(\mathcal{C}_i) - f_X(\mathcal{C}_{i+1})] \geq 0$ which implies that $E[f_X(\mathcal{C}_{i+1})] \leq E[f_X(\mathcal{C}_i)]$ and completes the proof. Note that if $E[Z \mid Z < \epsilon/7] > 0$ then we are done as we simply have a linear combination of two positive terms which is greater than 0. Let us focus on the case where $E[Z \mid Z < \epsilon/7] < 0$.

$$\begin{aligned}
&E[Z] = E[Z \mid Z \geq \epsilon/7]Pr[Z \geq \epsilon/7] + E[Z \mid Z < \epsilon/7]Pr[Z < \epsilon/7] \\
&\geq p\epsilon/7 + E[Z \mid Z < \epsilon/7](1 - p) \geq p\epsilon/7 - 4\gamma^2(1 - p) \\
&= (1 - O(1/n))\epsilon/7 - 4\gamma^2 O(\epsilon/\gamma^2 n) = (1 - O(1/n))\epsilon/7 - O(\epsilon/n) > 0
\end{aligned}$$

Where the first inequality is due to the definition of p and the fact that $E[Z \mid Z < \epsilon/7] < 0$, the second is due to the upper bound on f_X, and the last inequality is by assuming n is sufficiently large.

□

C FULL EXPERIMENTAL RESULTS

We list our full experimental results in this section. We use the β prefix to denote that the algorithm uses the learning rate of (Schwartzman, 2023). τ denotes the maximum number of data points used to represent each truncated cluster center. We investigate 3 kernel functions: 1) The Gaussian kernel, as presented in Section 6, 2) The k-nearest-neighbor (k-nn) kernel, where the kernel matrix is $D^{-1}AD^{-1}$, A is a k-nn adjacency matrix of the data and D is the corresponding degree matrix, and 3) the heat kernel (Chung, 1997) where the kernel matrix is $\exp(-tD^{-1/2}AD^{-1/2})$ for some $0 < t < \infty$, A is a k-nn adjacency matrix and D is the corresponding degree matrix. All parameter settings can be found in the supplementary material.

Unlike for the Gaussian kernel where $\gamma = 1$; We observe empirically that for both the k-nn and heat kernels, $\gamma \ll 1$. In this case, the dependence on $\max\{\gamma^4, \gamma^2\}$ in the batch size required for Theorem 1 actually helps us. We found the parameters for these kernels to be easier to tune in practise than the Gaussian kernel parameter σ. For each kernel, we recorded the empirical value of gamma as follows:

Dataset	**Kernel Type**	γ
pendigits	knn	0.00100
pendigits	heat	0.0477
pendigits	gaussian	1
har	knn	0.000500
har	heat	0.0468
har	gaussian	1
mnist_784	knn	0.00220
mnist_784	heat	0.0612
mnist_784	gaussian	1
letter	knn	0.00100
letter	heat	0.0399
letter	gaussian	1

Table 1: γ values for various datasets and kernel types, rounded to 3 significant figures.

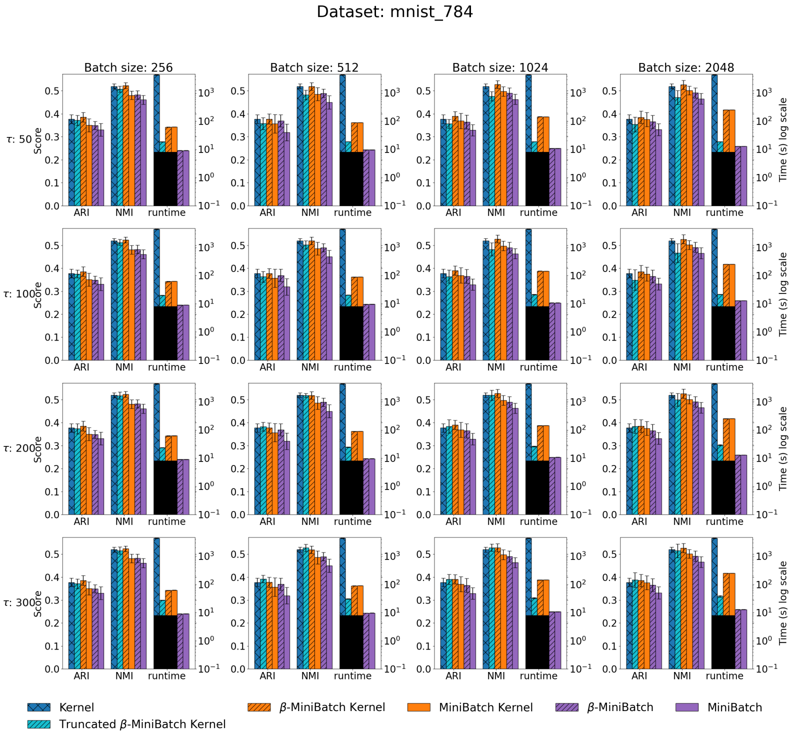

Figure 2: Experimental results on the MNIST dataset where the kernel algorithms use the Gaussian kernel.

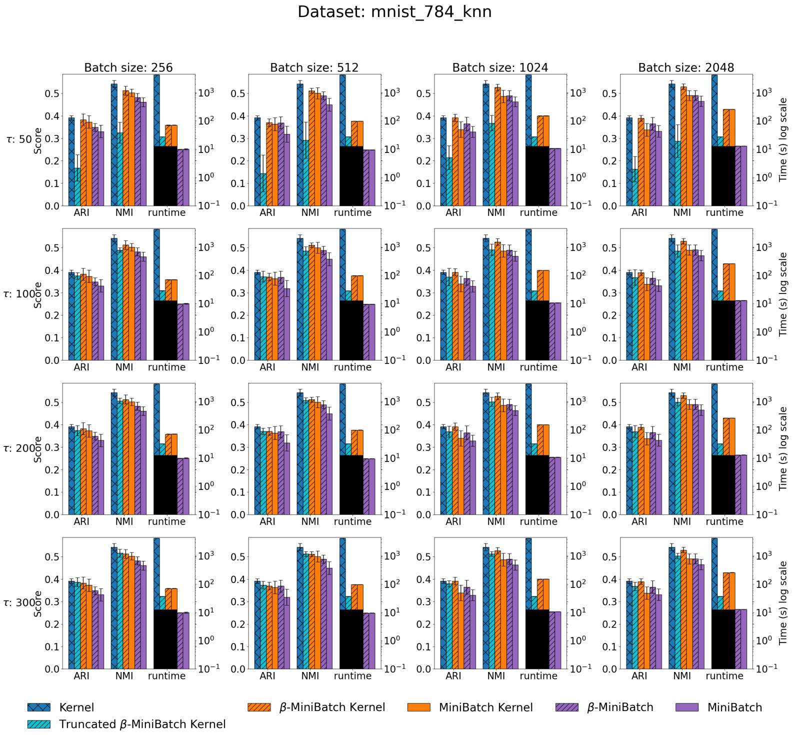

Figure 3: Experimental results on the MNIST dataset where the kernel algorithms use the k-nn kernel.

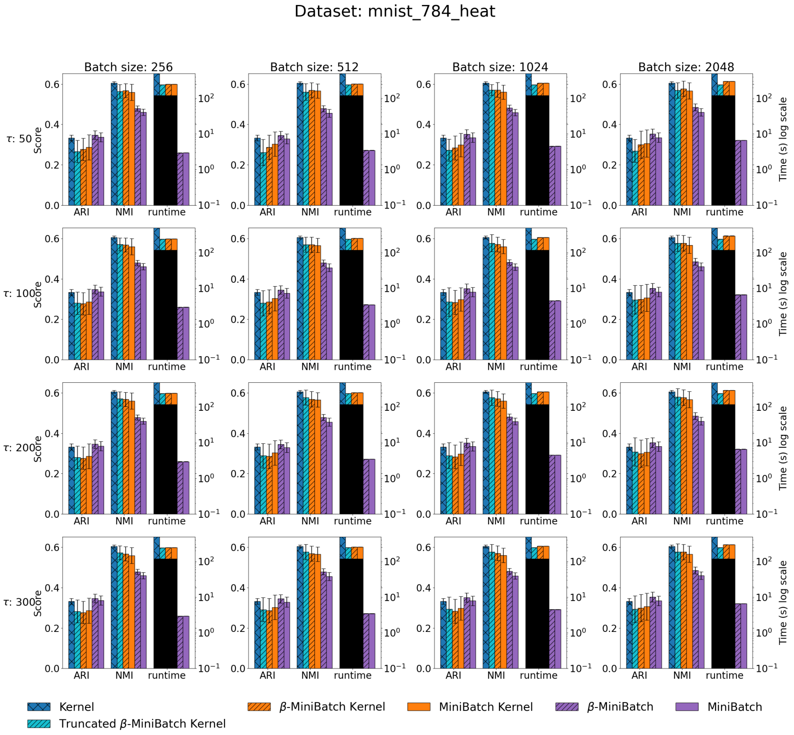

Figure 4: Experimental results on the MNIST dataset where the kernel algorithms use the Heat kernel.

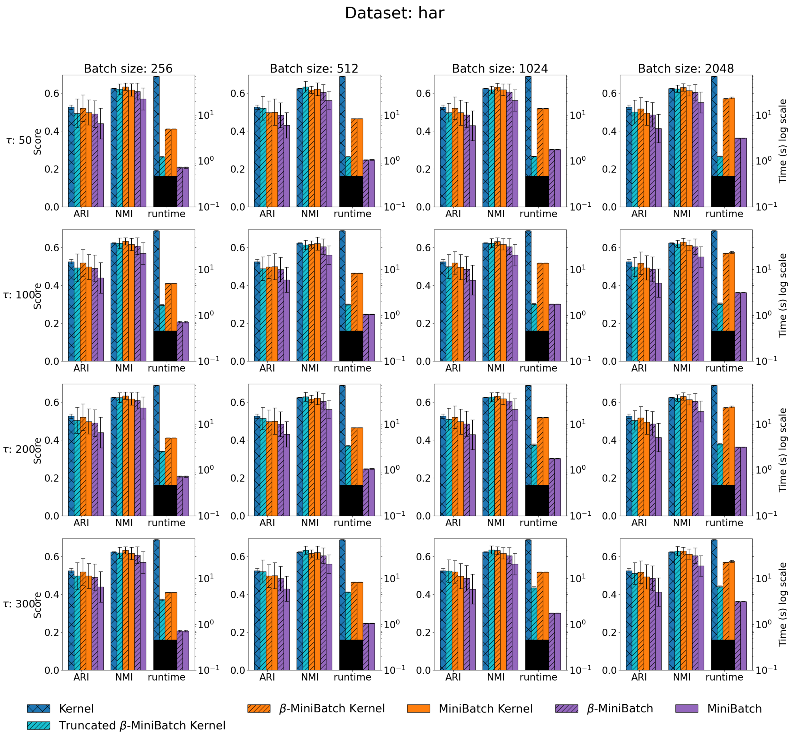

Figure 5: Experimental results on the Har dataset where the kernel algorithms use the Gaussian kernel.

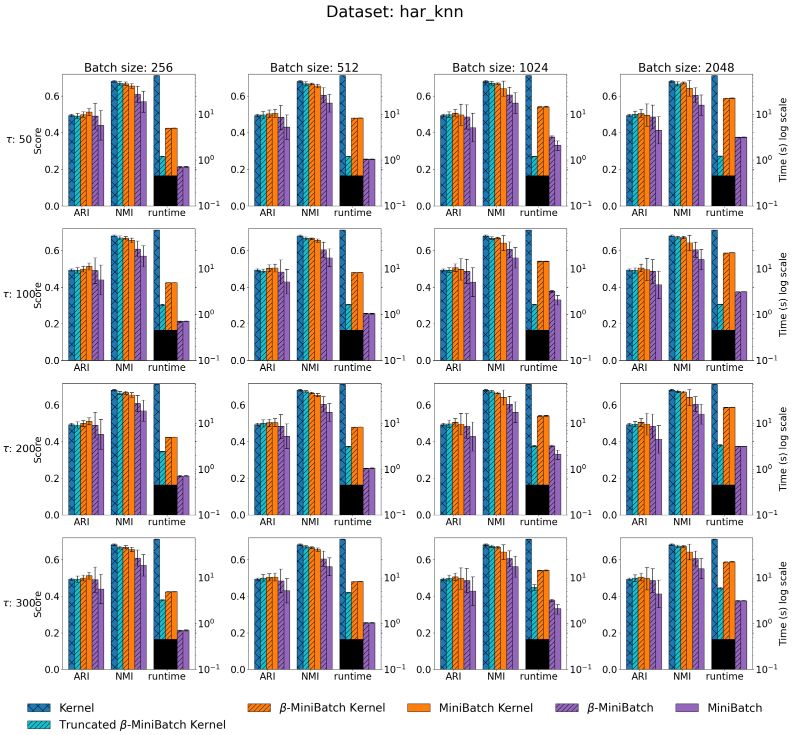

Figure 6: Experimental results on the Har dataset where the kernel algorithms use the k-nn kernel.

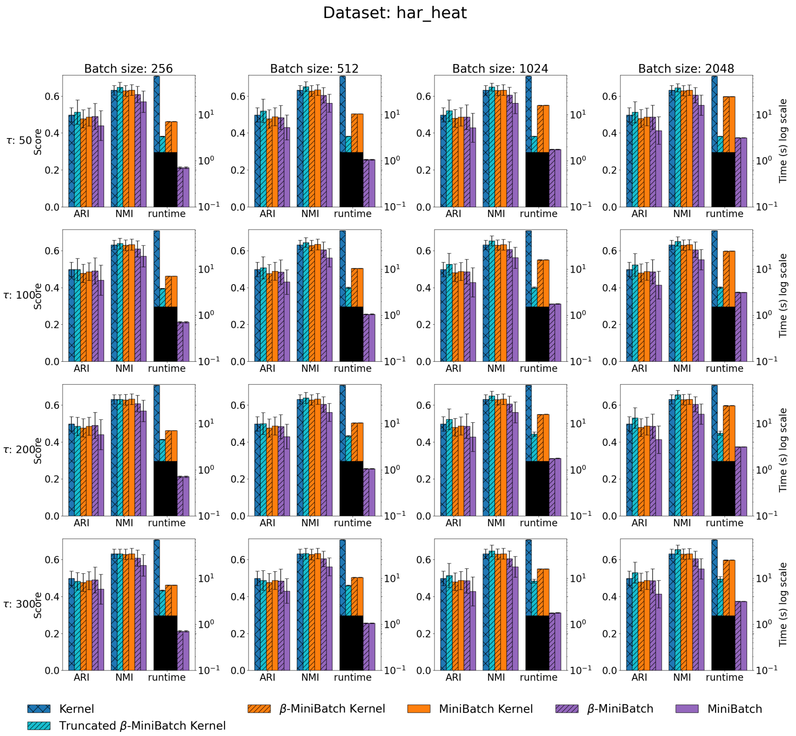

Figure 7: Experimental results on the Har dataset where the kernel algorithms use the Heat kernel.

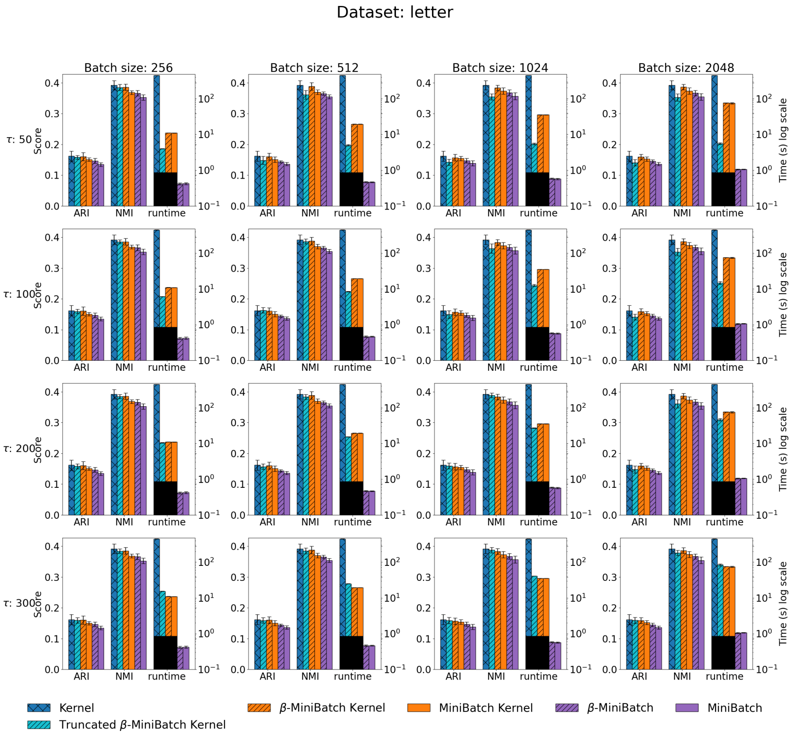

Figure 8: Experimental results on the Letter dataset where the kernel algorithms use the Gaussian kernel.

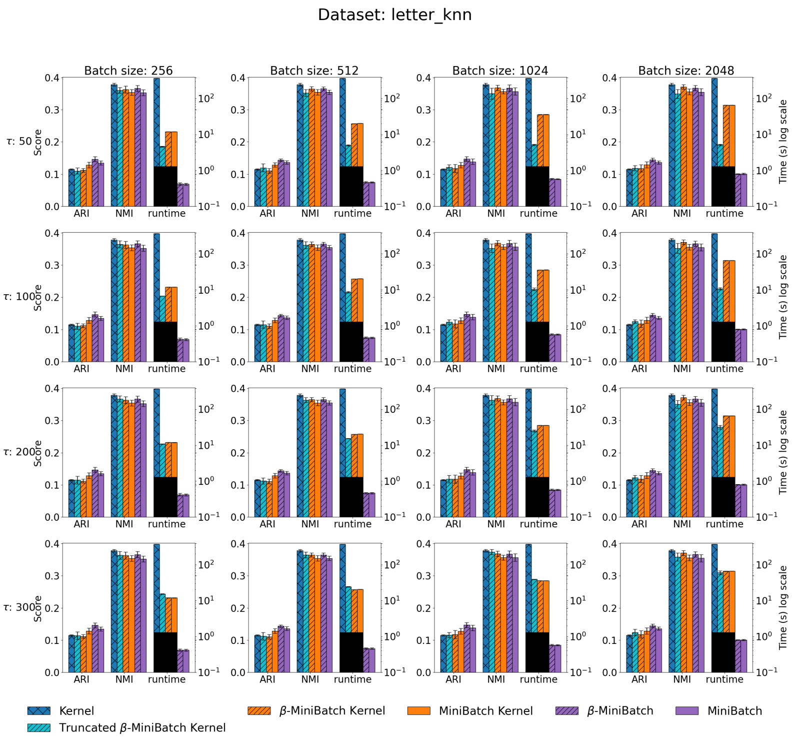

Figure 9: Experimental results on the Letter dataset where the kernel algorithms use the k-nn kernel.

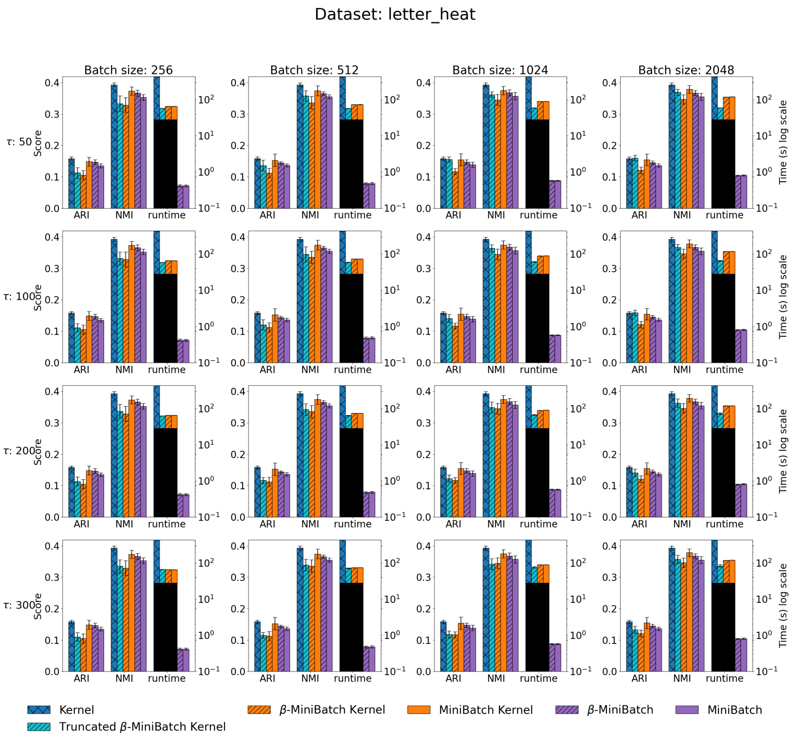

Figure 10: Experimental results on the Letter dataset where the kernel algorithms use the Heat kernel.

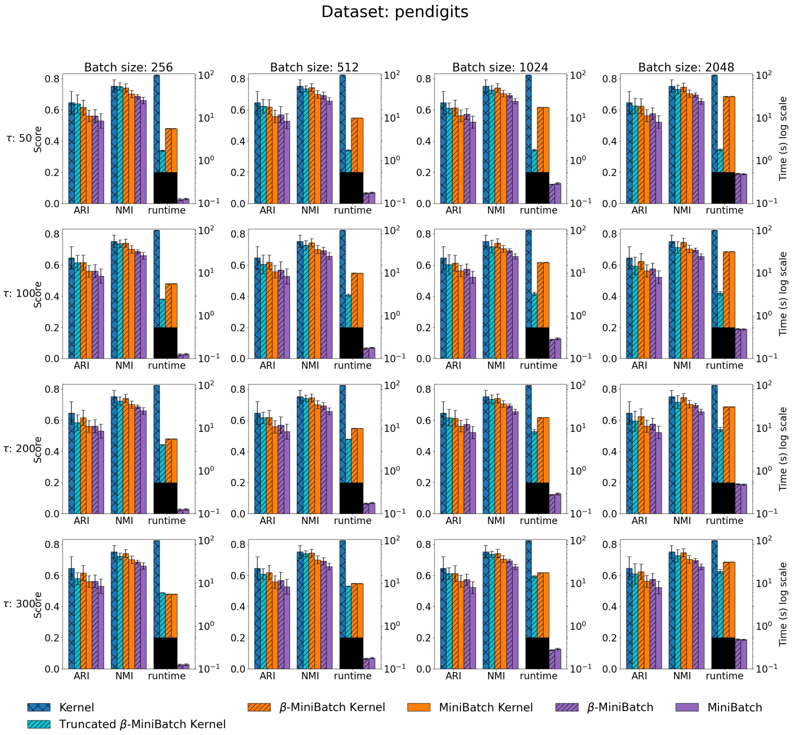

Figure 11: Experimental results on the Pendigits dataset where the kernel algorithms use the Gaussian kernel.

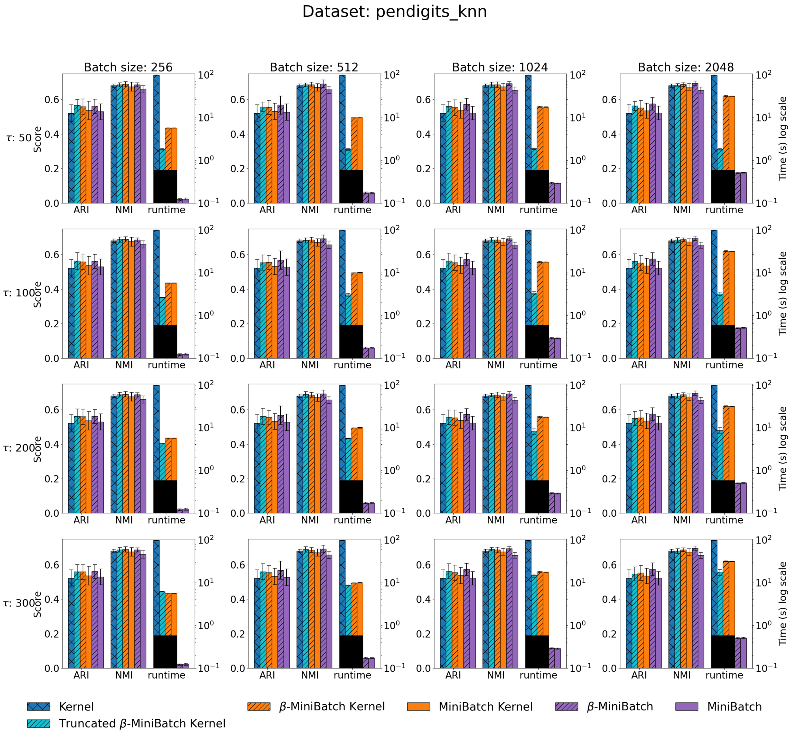

Figure 12: Experimental results on the Pendigits dataset where the kernel algorithms use the k-nn kernel.

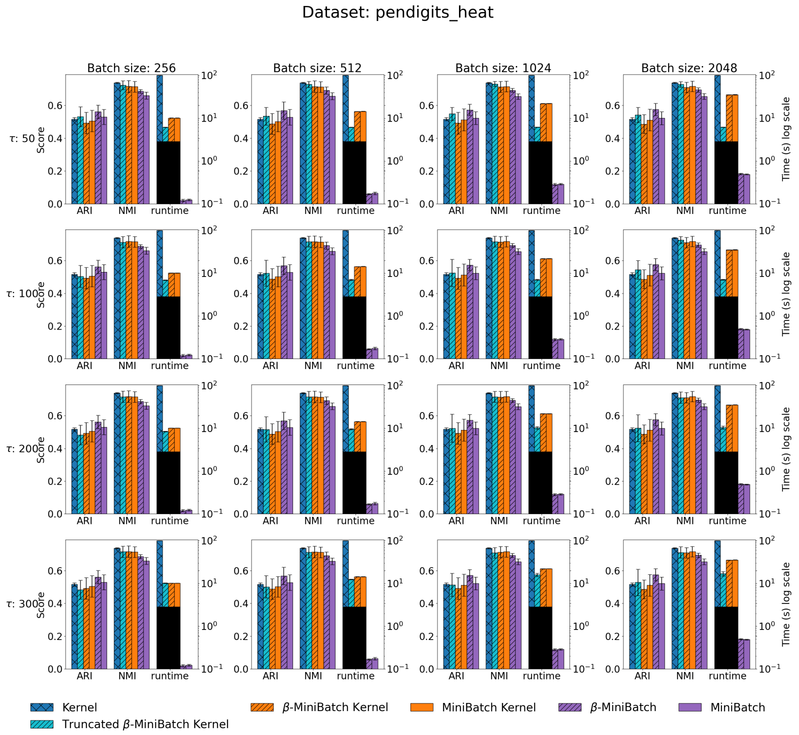

Figure 13: Experimental results on the Pendigits dataset where the kernel algorithms use the Heat kernel.