SparseKmeans: Efficient K-means Clustering For Sparse Data

Khoi Nguyen Pham Dang*
He-Zhe Lin*
khoinguyen.phamdang@mbzuai.ac.ae
b07902028@csie.ntu.edu.tw
Mohamed bin Zayed University of Artificial Intelligence
Abu Dhabi, UAE

Chih-Jen Lin
National Taiwan University
Taipei, Taiwan
Mohamed bin Zayed University of Artificial Intelligence
Abu Dhabi, UAE
cjlin@csie.ntu.edu.tw

ABSTRACT

We introduce SparseKmeans, the first Python package for fast K-means clustering on high-dimensional sparse data. Most existing K-means implementations, such as scikit-learn, are only optimized for dense data and do not run efficiently on sparse inputs. In this work, we thoroughly investigate how to accelerate widely used K-means algorithms on sparse data via matrix operations. In particular, we propose a new design of Elkan's method that aggregates distance computations and reduces fragmented memory access. By analyzing the structure of key matrices and leveraging highly optimized sparse matrix libraries, SparseKmeans achieves up to 9x speedup over scikit-learn. The package is available at https://github.com/cjlin1/sparsekmeans.

CCS CONCEPTS

• **Computing methodologies** → *Unsupervised learning*.

KEYWORDS

K-means algorithm; Elkan's method; Lloyd's method; unsupervised learning; clustering; sparse matrix operations

ACM Reference Format:
Khoi Nguyen Pham Dang, He-Zhe Lin, and Chih-Jen Lin. 2025. SparseKmeans: Efficient K-means Clustering For Sparse Data. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management (CIKM '25), November 10–14, 2025, Seoul, Republic of Korea.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3746252.3761646

1 INTRODUCTION

K-means is one of the most fundamental clustering techniques in machine learning and data mining. It is supported in many popular machine learning libraries such as scikit-learn [8], SciPy [10], and MLlib [7]. In particular, many Python users perform K-means with scikit-learn. However, while sparse data is prevalent in real-world applications, most packages – including SciPy and MLlib – support clustering only on dense matrix formats. Although scikit-learn has a specialized implementation for sparse K-means, its efficiency is not as well optimized as it is for dense inputs.[1] More importantly, there has been little research on implementing K-means algorithms specialized for sparse data, leading to substantial inefficiencies. As a result, the community has requested an efficient package for sparse K-means.[2,3]

In this work, we propose SparseKmeans, an easy-to-use Python package for efficient K-means clustering on high-dimensional sparse data sets. It supports all K-means methods available in scikit-learn, including Lloyd [6] and Elkan [4]. Internally, SparseKmeans novelly implements both methods using sparse matrix operations and leverages the highly optimized GraphBLAS library [2] for these operations. To the best of our knowledge, this is the first work to fully express all K-means steps as sparse matrix operations. On large-scale sparse data sets, SparseKmeans achieves up to 9x speedups over the K-means implementation in scikit-learn. Moreover, our implementation is simple and written entirely in pure Python, without the need for handcrafted C-level codes as used in scikit-learn. The package is available at https://github.com/cjlin1/sparsekmeans.

Paper organization. In Section 2, we introduce the K-means algorithm and its challenges on sparse data sets. In Section 3, we propose a novel matrix-based design of Elkan's method. We discuss the optimization of matrix operations to address the challenges mentioned earlier. We present the package in Section 4 and show the running time in Section 5. Conclusions are given in Section 6 and supplementary materials are provided at https://www.csie.ntu.edu.tw/~cjlin/papers/sparse_kmeans/supplementary.pdf.

2 K-MEANS AND CHALLENGES TO HANDLE SPARSE DATA

Given a set of samples $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\}$, $\boldsymbol{x}_i \in \mathbb{R}^n$, the K-means clustering aims to partition the m points into K clusters. The partitioning can be represented by a label vector $\boldsymbol{\ell} \in \{1, \ldots, K\}^m$, where ℓ_i is the cluster for $\boldsymbol{x}_i$. The objective of the K-means problem (with respect to the Euclidean norm) is

$$\min_{\boldsymbol{\ell}} \sum_{i=1}^{m} \|\boldsymbol{x}_i - \boldsymbol{c}_{\ell_i}\|, \tag{1}$$

where each $\boldsymbol{c}_j$ with $j \in \{1, \ldots, K\}$ is the centroid for cluster-j, defined by

$$\boldsymbol{c}_j = \frac{\sum_{i:\ell_i=j} \boldsymbol{x}_i}{(\text{size of cluster-}j)} = \frac{\sum_{i:\ell_i=j} \boldsymbol{x}_i}{|\{\boldsymbol{x}_i \mid \ell_i = j\}|}. \tag{2}$$

*Both authors contributed equally to this research.

CIKM '25, November 10–14, 2025, Seoul, Republic of Korea.

ACM ISBN 979-8-4007-2040-6/2025/11
https://doi.org/10.1145/3746252.3761646

[1] For dense data, beyond scikit-learn, there are other advanced libraries focusing on efficient clustering, such as [3].
[2] https://stackoverflow.com/questions/58346524/faster-kmeans-clustering-on-high-dimensional-data-with-gpu-support
[3] In [9], the authors mentioned "Another difficulty is a lack of efficient implementations for sparse k-means in dealing with large datasets."

Typically, problem (1) is solved with an iterative framework:

1. Choose K points as the initial centroids $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_K$.
2. Repeat the following until the stopping condition is met.[4]
   (a) *Assign-step:* For each $\boldsymbol{x}_i$, assign it to the nearest cluster
   $$\ell_i \in \arg\min_{j=1,\ldots,K} \|\boldsymbol{x}_i - \boldsymbol{c}_j\|. \tag{3}$$
   (b) *Update-step:* For each $j \in \{1, \ldots, K\}$, update $\boldsymbol{c}_j$ by (2).

The two well-known methods implemented in scikit-learn – Lloyd's [6] and Elkan's [4] – are within this framework. Their main difference lies in how they identify the nearest cluster in Step-2a. In the following subsections, we discuss the optimization challenges of each step when working with sparse inputs.

2.1 Step-2a: Cluster Assignment

Cluster assignment is the major computational bottleneck, regardless of Lloyd's or Elkan's methods. Both require many distance calculations between data points and centroids, i.e., those $\|\boldsymbol{x}_i - \boldsymbol{c}_j\|$'s in the Assign-step.

2.1.1 Lloyd's Method. According to (3), Lloyd's method first calculates the squared distance $\|\boldsymbol{x}_i - \boldsymbol{c}_j\|^2$ between all sample points and centroids, which is the following matrix in $\mathbb{R}^{m\times K}$

$$\begin{bmatrix} \|\boldsymbol{x}_1\|^2 - 2\boldsymbol{x}_1^T\boldsymbol{c}_1 + \|\boldsymbol{c}_1\|^2 & \cdots & \|\boldsymbol{x}_1\|^2 - 2\boldsymbol{x}_1^T\boldsymbol{c}_K + \|\boldsymbol{c}_K\|^2 \\ \|\boldsymbol{x}_2\|^2 - 2\boldsymbol{x}_2^T\boldsymbol{c}_1 + \|\boldsymbol{c}_1\|^2 & \cdots & \|\boldsymbol{x}_2\|^2 - 2\boldsymbol{x}_2^T\boldsymbol{c}_K + \|\boldsymbol{c}_K\|^2 \\ \vdots & \ddots & \vdots \\ \|\boldsymbol{x}_m\|^2 - 2\boldsymbol{x}_m^T\boldsymbol{c}_1 + \|\boldsymbol{c}_1\|^2 & \cdots & \|\boldsymbol{x}_m\|^2 - 2\boldsymbol{x}_m^T\boldsymbol{c}_K + \|\boldsymbol{c}_K\|^2 \end{bmatrix}. \tag{4}$$

After that, we take the minimum of each row to find the nearest cluster for each $\boldsymbol{x}_i$. To calculate (4), we need the squared norms $\|\boldsymbol{x}_i\|^2$, $\|\boldsymbol{c}_j\|^2$, which are relatively cheap, and the more expensive matrix-matrix multiplication

$$XC^T = \begin{bmatrix} - & \boldsymbol{x}_1^T & - \\ - & \boldsymbol{x}_2^T & - \\ & \vdots & \\ - & \boldsymbol{x}_m^T & - \end{bmatrix} \begin{bmatrix} | & | & & | \\ \boldsymbol{c}_1 & \boldsymbol{c}_2 & \cdots & \boldsymbol{c}_K \\ | & | & & | \end{bmatrix}. \tag{5}$$

For dense data (i.e., when X contains many non-zero entries), scikit-learn leverages optimized Basic Linear Algebra Subprograms (BLAS) to compute (5), significantly reducing the computation time. In contrast, handling sparse data introduces some challenges when applying efficient sparse matrix libraries:

- The performance of sparse matrix products depends on various factors such as storage formats and the way to do the operation. For example, one needs to decide whether to store X and C using a "row-major" or "column-major" format. Our benchmark results show that, for sparse X and C, computing (5) under a "column-major" C^T can be 60 times slower than under a "row-major" C^T. Moreover, the development for efficient sparse matrix operations is still less mature compared to that for dense matrices.
- Because $\boldsymbol{c}_j$ is often initialized to be some $\boldsymbol{x}_i$ in Step-1,[5] C is rather sparse at the first iteration. However, in later updates, the density of C depends on many factors such as the density of X, the number of clusters, and the partition result. Thus, optimizing (5) becomes difficult since the density of C is hard to predict.

[4] See supplementary materials for details.

[5] We use the Kmeans++ initialization [1] as scikit-learn does. See supplementary for details.

Algorithm 1: A sketch of Elkan's cluster assignment.

```
Compute D ∈ R^{K×K} with D_{j,j'} = ||c_j − c_{j'}||/2
for i = 1, ..., m do
    for j = 1, ..., K do
        // compute ||x_i − c_j|| when neither (6) nor (7) holds
        if j ≠ ℓ_i and D_{j,ℓ_i} < u(x_i) and l(x_i, c_j) < u(x_i)
         then
            l(x_i, c_j) ← ||x_i − c_j||
            if ||x_i − c_j|| < ||x_i − c_{ℓ_i}|| then
                ℓ_i ← j, u(x_i) ← ||x_i − c_j||
```

2.1.2 Elkan's Method. This method avoids some calculations of $\|\boldsymbol{x}_i - \boldsymbol{c}_j\|$'s by exploiting the information from the previous iteration. Specifically, the algorithm uses the triangle inequality to maintain two distance-related bounds.

- For each $\boldsymbol{x}_i$, let $u(\boldsymbol{x}_i)$ be an upper bound of $\|\boldsymbol{x}_i - \boldsymbol{c}_{\ell_i}\|$, i.e., the distance to the closest cluster-ℓ_i.
- For each $\boldsymbol{x}_i$ and any $j = 1, \ldots, K$, let $l(\boldsymbol{x}_i, \boldsymbol{c}_j)$ be a lower bound of $\|\boldsymbol{x}_i - \boldsymbol{c}_j\|$.

We leave the details of maintaining $u(\boldsymbol{x}_i)$ and $l(\boldsymbol{x}_i, \boldsymbol{c}_j)$ in the supplementary materials.

Let us see how these bounds effectively reduce the number of distance calculations. Suppose at the end of iteration-t, $\boldsymbol{x}_i$ belongs to cluster-ℓ_i, with the centroid being $\boldsymbol{c}_{\ell_i}$. To determine the cluster of $\boldsymbol{x}_i$ at iteration-$(t+1)$, Elkan's method uses the following two criteria to filter out clusters that $\boldsymbol{x}_i$ cannot belong to.

(1) For $j \neq \ell_i$ satisfying
$$u(\boldsymbol{x}_i) \leq \frac{1}{2}\|\boldsymbol{c}_j - \boldsymbol{c}_{\ell_i}\|, \tag{6}$$
the triangle inequality shows that $\boldsymbol{x}_i$ is closer to cluster-ℓ_i than cluster-j, so $\boldsymbol{x}_i$ must not fall into cluster-j:
$$\begin{aligned} \|\boldsymbol{x}_i - \boldsymbol{c}_j\| &\geq \|\boldsymbol{c}_{\ell_i} - \boldsymbol{c}_j\| - \|\boldsymbol{x}_i - \boldsymbol{c}_{\ell_i}\| \\ &\geq 2u(\boldsymbol{x}_i) - u(\boldsymbol{x}_i) \geq \|\boldsymbol{x}_i - \boldsymbol{c}_{\ell_i}\|. \end{aligned}$$

(2) For $j \neq \ell_i$ satisfying
$$u(\boldsymbol{x}_i) \leq l(\boldsymbol{x}_i, \boldsymbol{c}_j), \tag{7}$$
we have
$$\|\boldsymbol{x}_i - \boldsymbol{c}_{\ell_i}\| \leq u(\boldsymbol{x}_i) \leq l(\boldsymbol{x}_i, \boldsymbol{c}_j) \leq \|\boldsymbol{x}_i - \boldsymbol{c}_j\|,$$
showing that $\boldsymbol{x}_i$ must not belong to cluster-j.

For j satisfying either (6) or (7), we do not need to calculate $\|\boldsymbol{x}_i - \boldsymbol{c}_j\|$. Based on the two criteria, in Algorithm 1, we give a sketch of Elkan's cluster assignment within one K-means iteration.[6]

At first glance, it seems possible to implement Algorithm 1 by aggregating the distance calculations
$$\|\boldsymbol{x}_i - \boldsymbol{c}_j\| \quad \forall j \text{ not satisfying (6) and (7)}, \tag{8}$$
which requires only one matrix-vector product. Unfortunately, the procedure in Algorithm 1 is inherently sequential. Specifically, in

[6] In fact, Elkan [4] applies more complicated settings. For better understanding, here we only show the main idea and leave the details to supplementary materials.

Line 6, both ℓ_i and $u(\boldsymbol{x}_i)$ may be updated when $\boldsymbol{x}_i$ is found to be closer to cluster-j. As a result, in subsequent iterations over j, we need to evaluate conditions (6) and (7) using the most recent values of ℓ_i and $u(\boldsymbol{x}_i)$. This dependency hinders the aggregation of several $\|\boldsymbol{x}_i - \boldsymbol{c}_j\|$'s, as it is impossible to determine which distances will be needed in advance.

2.2 Step-2b: Centroid Updates

From (2), since each $\boldsymbol{c}_j$ is a linear combination of $\boldsymbol{x}_i$'s, so we can compute the centroids by the following matrix product

$$C = \begin{bmatrix} - & \boldsymbol{c}_1^T & - \\ - & \boldsymbol{c}_2^T & - \\ & \vdots & \\ - & \boldsymbol{c}_K^T & - \end{bmatrix} = BX = \begin{bmatrix} B_{11} & \cdots & B_{1m} \\ B_{21} & \cdots & B_{2m} \\ \vdots & \ddots & \vdots \\ B_{K1} & \cdots & B_{Km} \end{bmatrix} \begin{bmatrix} - & \boldsymbol{x}_1^T & - \\ - & \boldsymbol{x}_2^T & - \\ & \vdots & \\ - & \boldsymbol{x}_m^T & - \end{bmatrix}, \tag{9}$$

where $B \in \mathbb{R}^{K \times m}$ is defined with

$$B_{ji} = \begin{cases} 1/(\text{size of cluster-}j) & \text{if } \ell_i = j \\ 0 & \text{otherwise} \end{cases}. \tag{10}$$

There are two challenges for handling (9). First, as discussed in Section 2.1.1 the density of C remains unknown until (9) is actually computed. Therefore, it is hard to decide in advance whether C should be stored in a dense or sparse format. Second, even if X is extremely sparse and we decide to store C as a sparse matrix, it is well known that the sparse-sparse matrix product (9) is very difficult due to the irregular structure of the output. Several studies, such as [11], have investigated the efficient implementation to have a sparse output matrix C, but the appropriate storage format for C remains uncertain.

3 ACCELERATING K-MEANS ALGORITHM VIA EFFICIENT SPARSE MATRIX OPERATIONS

The core idea of SparseKmeans is to leverage highly optimized packages for sparse matrix operations. For Lloyd's method, the key computations – namely (5) and (9) – are already expressed as matrix multiplications. However, as noted in Section 2.1.2, the sequential procedure in Algorithm 1 makes it difficult to apply such operations to Elkan's method. Therefore, in Section 3.1, we first introduce a new design of Elkan's method, which well aggregates the distance calculations by using matrix-vector products. Then, in Section 3.2, we further discuss the practical issues of applying existing libraries to support these matrix computations effectively.

3.1 A New Matrix-based Design of Elkan's Method

The main obstacle that prevents Algorithm 1 from utilizing matrix operations is the "sample-wise" cluster assignment. The sequential procedure within the assignment of an $\boldsymbol{x}_i$ makes it impossible to compute (8) together. To address this issue, we redesign Elkan's method in a "cluster-wise" manner, as shown in Algorithm 2. Specifically, for each cluster-j, our new design first finds the set of candidate samples satisfying neither (6) nor (7), denoted by S_j:

$$S_j = \{i \mid \ell_i \neq j, \|\boldsymbol{c}_j - \boldsymbol{c}_{\ell_i}\|/2 < u(\boldsymbol{x}_i) \text{ and } l(\boldsymbol{x}_i, \boldsymbol{c}_j) < u(\boldsymbol{x}_i)\}. \tag{11}$$

Algorithm 2: A new design of Elkan's cluster assignment that allows vectorization.

```
Compute D ∈ R^{K×K} with D_{j,j'} = ‖c_j − c_{j'}‖/2
Compute d ∈ R^m with d_i = ‖x_i − c_{ℓ_i}‖
for j = 1, ..., K do
    for i = 1, ..., m do
        if j ≠ ℓ_i and D_{j,ℓ_i} < d_i and l(x_i, c_j) < d_i then
            l(x_i, c_j) ← ‖x_i − c_j‖
            if d_i > ‖x_i − c_j‖ then
                ℓ_i ← j, d_i ← ‖x_i − c_j‖
```

Under a fixed j, checking whether each $\boldsymbol{x}_i \in S_j$ is independent to one another, so we can vectorize the comparisons for all $\boldsymbol{x}_i$'s. After having S_j, we aggregate the calculation of $\|\boldsymbol{x}_i - \boldsymbol{c}_j\|$ for all $i \in S_j$, whose major computation is the matrix-vector product

$$X_{S_j,:} \cdot \boldsymbol{c}_j, \tag{12}$$

where $X_{S_j,:}$ denotes a submatrix of X containing the rows of samples in S_j. This new design successfully bypasses the sequential restriction in Algorithm 1.

On the other hand, readers may find that when checking (6) and (7), we replace $u(\boldsymbol{x}_i)$ with d_i, the exact $\|\boldsymbol{x}_i - \boldsymbol{c}_{\ell_i}\|$ (which serves as the tightest upper bound). By using the smallest possible values in the left-hand side of (6) and (7), the number of needed distance calculations is reduced. While maintaining $\boldsymbol{d}$ could induce additional costs, it turns out to be necessary for handling empty clusters during centroid updates, as done in scikit-learn (see supplementary for details). As a result, $\boldsymbol{d}$ is already maintained, and we can reuse it without incurring additional computation.

Besides the matrix-vector products (12), there are two major computations in Algorithm 2:

- We need to calculate CC^T for precomputing the pairwise centroid distance matrix D.
- The vector $\boldsymbol{d}$: From

$$d_i = \|\boldsymbol{x}_i - \boldsymbol{c}_{\ell_i}\| = \sqrt{\|\boldsymbol{x}_i\|^2 - 2\boldsymbol{x}_i^T \boldsymbol{c}_{\ell_i} + \|\boldsymbol{c}_{\ell_i}\|^2},$$

  we need $\|\boldsymbol{x}_i\|^2$, $\|\boldsymbol{c}_{\ell_i}\|^2$ and $\boldsymbol{x}_i^T \boldsymbol{c}_{\ell_i}$ for all i. Among the three terms, $\|\boldsymbol{x}_i\|$ is only calculated once over the K-means algorithm. To have $\|\boldsymbol{c}_{\ell_i}\|^2$, we only need to compute K squared norms $\|\boldsymbol{c}_j\|^2, \forall j$. Since $m \gg K$, the bottleneck lies in the m inner products $\boldsymbol{x}_i^T \boldsymbol{c}_{\ell_i}$.

3.2 Optimization of Sparse Matrix Products

Up to now, there are five matrix/vector operations involved in K-means algorithms:

(I) The matrix-matirx product XC^T in (5) of Lloyd's method.
(II) The matrix-matirx product CC^T in Elkan's method.
(III) The inner products $\boldsymbol{x}_i^T \boldsymbol{c}_{\ell_i}, \forall i$ in Elkan's method.
(IV) The matrix-vector product $X_{S_j,:}\boldsymbol{c}_j$ in (12) of Elkan's method.
(V) The matrix-matirx product $C = BX$ in (9) for centroid updates.

We choose the state-of-the-art package SuiteSparse:GraphBLAS [2], which supports various kinds of sparse matrix operations, to compute these operations. On a wide range of tasks, SuiteSparse:GraphBLAS outperforms other sparse matrix libraries such

Table 1: The running time comparison (in seconds) for Lloyd's and Elkan's methods using scikit-learn (sklearn) and SparseKmeans. The best setting for each dataset and K is shown in bold. We terminate the program if it takes more than 10^5 seconds.

Data sets	# Samples (m)	# Features (n)	K	Lloyd's method			Elkan's method		
				sklearn	SparseKmeans	Speedup	sklearn	SparseKmeans	Speedup
Wiki-500K	500,091	2,381,304	100	24,617.7	2,957.3	8.32x	4,042.7	**2,382.1**	1.69x
	(Density: ≈ 0.10%)		500	$> 10^5$	26,104.9	> 3.83x	91,441.8[7]	**5,061.0**	18.06x
Amazon-670K	667,317	135,910	100	670.9	**123.4**	5.43x	248.5	141.8	1.75x
	(Density: ≈ 0.16%)		500	7,170.1	776.2	9.23x	1,248.3	**685.1**	1.82x
Url	2,396,130	3,231,962	100	799.9	**163.5**	4.89x	719.8	296.0	2.43x
	(Density: ≈ 0.003%)		500	5,888.4	**987.2**	5.96x	4,687.1	1,989.7	2.35x
Amazon-3M	2,812,281	337,067	100	26,207.5	2,359.1	11.10x	2,965.2	**1,743.4**	1.70x
	(Density: ≈ 0.15%)		500	$> 10^5$	39,346.1	> 2.54x	13,340.0	**5,517.9**	2.41x

as Intel MKL [5]. To make the best use of the package, we painstakingly compare the running time under different matrix formats (e.g., row/column-major, dense/sparse) to determine the most suitable configuration for the five key operations. Below, we summarize the final settings used in SparseKmeans.

For matrix products that do not require extracting specific rows or columns from X or C, including operations (I), (II), and (V), we always perform the operation using the best "row-major × row-major" setting. The most difficult part is to decide whether C should be stored in a dense or sparse format. Although it is impossible to know the density of C in advance, empirically we observe that the density of C is close to that in the previous iteration. Therefore, we rely on the density of C from the previous iteration and set a heuristic threshold $\gamma = 0.1$ to choose the storage format for C:

- If the density of C in the previous iteration is below γ, we use a sparse format to store the output of (V) and conduct (I) and (II).
- Otherwise, a dense format for C is applied.

Operations (III) and (IV) require accessing specific rows/columns from X or C. Interestingly, our study and benchmark results indicate that a dense C (or $\boldsymbol{c}_{\ell_i}$ and $\boldsymbol{c}_j$ in the operations) should be used to achieve better performance (details provided in supplementary materials).

4 THE PACKAGE

Currently, SparseKmeans supports two formats for sparse inputs – `scipy.sparse.csr_matrix` and `graphblas.Matrix`. Running the package is straightforward and requires only a few lines of code:

```
kmeans = LloydKmeans(n_clusters=100) # Lloyd's K-means
kmeans = ElkanKmeans(n_clusters=100) # Elkan's K-means
labels = kmeans.fit(X)
```

The two classes `LloydKmeans` and `ElkanKmeans` inherit from a shared base class, `SparseKmeans`, which encapsulates common functionalities such as the centroid updates. For most cases, SparseKmeans produces clustering results identical to those from scikit-learn. The only exception arises when the number of distinct points is smaller than the number of clusters K, in which case our implementation yields a more reasonable clustering outcome (see supplementary materials for details).

To pursue high performance, scikit-learn relies on low-level implementations such as Cython. In contrast, SparseKmeans is written entirely in Python, yet delivers faster runtime performance in many scenarios. Our implementation requires only one-third as many lines of code compared to the corresponding part in scikit-learn, demonstrating both simplicity and efficiency. Additionally, we provide detailed documentation of the implementation to help users understand the internal workings of the algorithms.

5 EXPERIMENTS

In this section, we compare the running time of our package, SparseKmeans, against scikit-learn on four sparse data sets. We give data statistics in Table 1 and provide more details in supplementary materials. We run experiments on an Intel Core i7-6900K CPU @ 3.20GHz using 16 threads. To ensure a fair comparison, we configure both packages with same random seeds and stopping conditions.[8]

Table 1 gives the running time for Lloyd's and Elkan's methods under $K = 100$ and $K = 500$. SparseKmeans consistently achieves significant speedups over scikit-learn across all data sets. Notably, the speedups for $K = 500$ are generally greater than those for $K = 100$. The reason might be that the centroid matrix C becomes sparser as K increases, allowing our efficient handling of sparse matrix operations to provide more advantages over scikit-learn.

Now, let us compare Lloyd's and Elkan's methods with SparseKmeans. In most settings, Elkan's method is faster than Lloyd's because the former requires fewer distance calculations. However, for the highly sparse data set Url, Lloyd's method is faster than Elkan's. Under the extremely sparse scenario, the distance calculation can be very fast, so Lloyd's method is still efficient even if it computes all distances in (4). In contrast, Elkan's method is more complicated and may induce some overhead.

6 CONCLUSIONS

In this work, we developed SparseKmeans, a highly efficient package for K-means clustering on sparse data. We studied the formulation of widely used K-means algorithms into matrix operations and carefully chose the most suitable settings when applying sparse matrix libraries. Building on this foundation, a promising direction

[7] The running time is supposed to be less than this number. However, it seems that there exists a problem with scikit-learn's implementation.

[8] For Amazon-670K, the needed iterations for convergence is different on scikit-learn and SparseKmeans due to numerical imprecision. Therefore, we run the iterations up to be the minimum of the two and report the results.

for future work is to extend our matrix-based approach on GPUs to further improve the performance.

ACKNOWLEDGMENTS

This work was supported in part by National Science and Technology Council of Taiwan grant NSTC-113-2222-E-002-005-MY3. We thank Tim Davis, the author of SuiteSparse:GraphBLAS, for providing guides of using the package.

GENAI USAGE DISCLOSURE

In this work, GenAI softwares are used only to improve the wording of existing text and to correct spelling and grammar. No part of the technical content, experiments, or analysis was generated by GenAI.

REFERENCES

[1] David Arthur and Sergei Vassilvitskii. 2007. k-means++: the advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. 1027–1035.

[2] Timothy A. Davis. 2019. Algorithm 1000: SuiteSparse:GraphBLAS: Graph Algorithms in the Language of Sparse Linear Algebra. *ACM Trans. Math. Software* 45, 4 (2019).

[3] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The Faiss library. (2024). arXiv:2401.08281 [cs.LG]

[4] Charles Elkan. 2003. Using the triangle inequality to accelerate k-means. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning (ICML)*. 147–153.

[5] Intel. 2025. *Intel® Math Kernel Library Reference Manual*. https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl-documentation.html

[6] Stuart Lloyd. 1982. Least Squares Quantization in PCM. *IEEE Transactions on Information Theory* 28, 2 (1982), 129–137.

[7] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Zadeh, Matei Zaharia, and Ameet Talwalkar. 2016. MLlib: Machine Learning in Apache Spark. *Journal of Machine Learning Research* 17, 1 (2016), 1235–1241.

[8] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: machine learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[9] Yifan Qiao, Parker Carlson, Shanxiu He, Yingrui Yang, and Tao Yang. 2024. Threshold-driven pruning with segmented maximum term weights for approximate cluster-based sparse retrieval. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 19742–19757.

[10] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C. J. Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020), 261–272.

[11] Jiong Zhang, Yau-Shian Wang, Wei-Cheng Chang, Wei Li, Jyun-Yu Jiang, Cho-Jui Hsieh, and Hsiang-Fu Yu. 2023. Build Faster with Less: A Journey to Accelerate Sparse Model Building for Semantic Matching in Product Search. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*. 4960–4966.