Sparse Kernel Clustering of Massive High-Dimensional Data sets with Large Number of Clusters

Radha Chitta, Anil K. Jain, Rong Jin
Dept. of Computer Science and Engineering
Michigan State University
East Lansing, MI, 48824, USA
chittara@msu.edu, jain@cse.msu.edu, rongjin@cse.msu.edu

ABSTRACT

In clustering applications involving documents and images, in addition to the large number of data points (N) and their high dimensionality (d), the number of clusters (C) into which the data need to be partitioned is also large. Kernel-based clustering algorithms, which have been shown to perform better than linear clustering algorithms, have high running time complexity in terms of N, d and C. We propose an efficient *sparse kernel k-means clustering algorithm*, which incrementally samples the most informative points from the data set using importance sampling, and constructs a sparse kernel matrix using these sampled points. Each row in this matrix corresponds to a data point's similarity with its p-nearest neighbors among the sampled points ($p \ll N$). This sparse kernel matrix is used to perform clustering and obtain the cluster labels. This combination of sampling and sparsity reduces both the running time and memory complexity of kernel clustering. In order to further enhance its efficiency, the proposed algorithm projects the data on to the top C eigenvectors of the sparse kernel matrix and clusters these eigenvectors using a modified k-means algorithm. The running time of the proposed sparse kernel k-means algorithm is linear in N and d, and logarithmic in C. We show analytically that only a small number of points need to be sampled from the data set, and the resulting approximation error is well-bounded. We demonstrate, using several large high-dimensional text and image data sets, that the proposed algorithm is significantly faster than classical kernel-based clustering algorithms, while maintaining clustering quality.

Categories and Subject Descriptors

I.5.3 [**Pattern Recognition**]: Clustering—*Algorithms*

General Terms

Algorithms

PIKM'15, October 23, 2015, Melbourne, Australia.

ACM 978-1-4503-3782-3/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2809890.2809896.

Keywords

Big data; Kernel clustering; Importance sampling; Sparsity

1. INTRODUCTION

An increasing amount of digital data is being generated in the form of text, images, audio and video through social networks, blogs, online transactions, smartphone sensors, etc [1]. Analysis of this massive amount of data can lead to interesting findings about users and their behavior patterns, which would be useful in making important business decisions. Clustering is one of the principal tools to analyze and organize data, with minimal supervision from domain experts. Many algorithms have been published in the literature to efficiently cluster large high-dimensional data sets [2–4]. Most of these algorithms are "linear" in nature, i.e. they assume that the data is linearly separable in the input space, and use measures such as the Euclidean distance to define the inter-point similarity. Kernel-based clustering algorithms such as kernel k-means [5], spectral clustering [6], support vector clustering [7], and maximum margin clustering [8], on the other hand, project the data into a high (possibly infinite) dimensional space, where the data is likely to be separable. By using non-linear similarity measures to define the inter-point similarity, they achieve higher clustering quality than linear clustering algorithms on real-world data sets. However, the running time complexity of the kernel-based clustering algorithms is at least $O(N^2d + N^2C)$, where N is the number of points in the data set, d represents its dimensionality and C represents the number of clusters (See Table 1).

Document and image data sets, containing millions of unlabeled high-dimensional points, usually belong to a large number of clusters. Finding clusters in such data sets is computationally expensive using kernel-based clustering techniques. Our aim is to speed up kernel-based clustering for data sets with large N, d and C. We focus on the kernel k-means algorithm due to its comparable performance with other kernel-based clustering algorithms [9, 10] and its simplicity. We present an online kernel clustering algorithm, called the *sparse kernel k-means algorithm*, which can efficiently cluster data sets with millions of points and hundreds of features into thousands of clusters, with significantly lower processing and memory requirements, and high clustering accuracy.

Approximate kernel clustering algorithms such as [2, 3] reduce the running time of kernel clustering by uniformly sampling an m-sized subset of the data and constructing a low-rank approximate kernel matrix using the sampled data.

These approaches reduce the running time complexity of kernel clustering to $O(Nmd + NmC)$. The number of samples m required to obtain a good approximation is dependent on the rank of the kernel matrix, which in turn depends on the number of clusters [11]. Clustering data sets with large number of clusters using these algorithms still requires sampling $O(N)$ number of points to sufficiently represent all the clusters. This renders the approximate kernel clustering algorithms infeasible for large N.

The proposed sparse kernel k-means algorithm reduces the running time and memory complexity of kernel clustering using two key ideas: (i) kernel approximation using incremental importance sampling, and (ii) kernel sparsity. Importance sampling involves selecting data points based on their novelty, measured in terms of statistical leverage scores [12]. However, finding the statistical leverage scores for the entire data involves computing the eigenvectors of the full $N \times N$ kernel matrix, which is computationally prohibitive for large N [13]. We design an efficient online method to sample the data based on their importance, thereby reducing the time required for sampling.

We also reduce the complexity of kernel computation and clustering by using sparsification. We compute the p-nearest neighbor graph (where p is a user-defined parameter) for the sampled points and use this sparse kernel matrix to obtain the cluster centers. Clustering is performed efficiently by first projecting the sampled data into a subspace spanned by the top eigenvectors of the sparse kernel matrix, and then clustering the projected points using a modified k-means algorithm, which uses randomized kd-trees (also denoted as k-d trees) [14] to find the nearest cluster center for each data point.

The runtime complexity of the proposed algorithm is linear in N and d, and logarithmic in C. We show that only a small subset ($m = \Omega(C \log C)$) of the data needs to be sampled, thereby reducing the memory requirements. We demonstrate empirically using several benchmark data sets that the proposed clustering algorithm is scalable to data sets containing millions of high-dimensional data points and thousands of clusters.

The rest of the article is organized as follows. In Section 2, we describe the kernel k-means algorithm, and some of the related work on enhancing the scalability of kernel-based clustering. We then discuss the two primary components of the proposed sparse kernel k-means algorithm: importance sampling and kernel sparsity. In Section 3, we outline our algorithm and analyze its complexity. Finally, in Sections 4 and 5, we present the results of our empirical analysis, and conclude the study with future work.

2. BACKGROUND

We first outline the kernel k-means algorithm, which forms the basis of the proposed sparse kernel k-means algorithm. We then describe importance sampling and kernel sparsity, the two tools employed by the proposed algorithm to reduce the running time complexity of kernel clustering.

Kernel k-means. The key principle behind kernel k-means is to project the data to a high-dimensional space using a non-linear function $\phi(\cdot)$ and execute k-means on the projected data. Given an input data set $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, $\mathbf{x}_i \in \Re^d$, to be clustered into C clusters, a user-defined non-linear symmetric positive semi-definite (SPSD) similarity function $\kappa(\cdot, \cdot)$, where $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$ is used to define the similarity between data points. The C clusters are obtained by minimizing the sum-of-squared-errors in the high-dimensional kernel space:

$$\min_{U \in \{0,1\}^{C \times N}} \max_{\{c_k(\cdot) \in \mathcal{H}_\kappa\}_{k=1}^{C}} \sum_{k=1}^{C} \sum_{i=1}^{N} U_{ki} \, ||c_k(\cdot) - \kappa(\mathbf{x}_i, \cdot)||^2_{\mathcal{H}_\kappa}, \quad (1)$$

which can be relaxed to the following trace maximization problem [10]:

$$\max_{U \in \{0,1\}^{C \times N}} \operatorname{tr}(\widetilde{U} K \widetilde{U}^\top). \quad (2)$$

In equations (1) and (2), $\mathcal{H}_\kappa$ represents the Reproducing Kernel Hilbert Space (RKHS) induced by $\kappa(\cdot, \cdot)$, $||\cdot||_{\mathcal{H}_\kappa}$ is the functional norm for $\mathcal{H}_\kappa$, K represents the $N \times N$ pairwise similarity matrix, defined by $K_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$, $c_k(\cdot)$ represents the k^{th} cluster center in the RKHS, U represents the $C \times N$ cluster membership matrix where $U_{kt} = 1$ if $\mathbf{x}_t$ belongs to the k^{th} cluster and 0 otherwise, and $\widetilde{U} = \operatorname{diag}(U\mathbf{1})^{-1/2} U$.

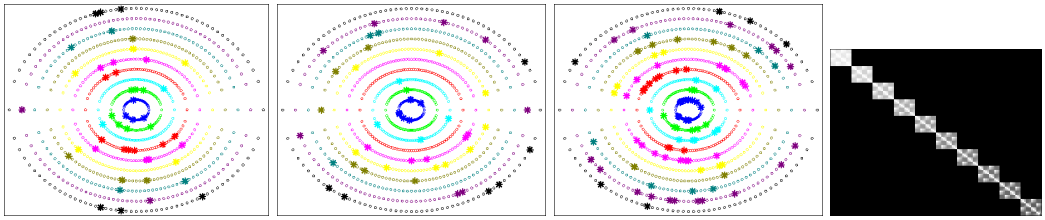
The complexity of the kernel k-means algorithm is $O(N^2 d + N^2 C)$, which makes it infeasible for data sets with millions of points. Other kernel-based clustering algorithms also have similar running time complexity.

Table 1: Complexity of popular partitional clustering algorithms. N and d represent the size and dimensionality of the data, respectively, and C represents the number of clusters. Parameter $m > C$ represents the size of the sampled subset for the sampling-based approximate clustering algorithms. $n_{sv} \geq C$ represents the number of support vectors. DBSCAN is dependent on user-defined intra-cluster and inter-cluster distance thresholds, so its complexity is not directly dependent on C.

Clustering algorithms	**Complexity**
k-means [15]	$O(NCd)$
DBSCAN [16]	$O(N \log(N)\, d)$
Kernel k-means [5]	$O(N^2 d + N^2 C)$
Spectral clustering [6]	$O(N^2 d + N^3 + NC^2)$
Support vector clustering [7]	$O(N^2 d n_{sv})$
Approximate spectral clustering [3]	$O(Nmd + NmC)$
Approximate kernel k-means [2]	$O(Nmd + NmC)$

A large body of literature has focused on reducing the running time and memory requirements of kernel-based clustering. Dimensionality reduction techniques such as low-rank embedding and random projection have been employed to cluster high-dimensional data sets [17]. In [18], the data was explicitly projected to the space spanned by a random subset of the data, using a special non-linear transformation and clustered in parallel over the Map-Reduce framework [19]. Approximate kernel-based clustering algorithms, such as Nystrom spectral clustering [3] and approximate kernel k-means [2], uniformly sample m points from the data set ($m \ll N$), construct a low-rank matrix using the sampled points, and obtain the cluster labels using this approximate kernel matrix. Both algorithms have running time complexities $O(Nmd + NmC)$, and require the sample size m to be greater than $\mu C \log C$ [11], where μ represents the coherence of the top C-dimensional eigenspace of K. When the number of clusters is large (say $O\left(\sqrt{N}\right)$), the number of samples required is $O(N)$, rendering these algorithms infeasible for large data sets. Our clustering algorithm reduces the minimum sample size by using importance sampling instead of uniform sampling.

Importance Sampling aims to select a subset of the data that is most informative. Let the kernel matrix K be decomposed as $K \simeq V_C \Sigma_C V_C^\top$, where $\Sigma_C = \operatorname{diag}(\lambda_1, \ldots, \lambda_C)$ contains the highest C eigenvalues of K and $V_C = (\mathbf{v}_1, \ldots, \mathbf{v}_C)$ contains the corresponding eigenvectors. A data point $\mathbf{x}_i$ is



(a) (b) (c) (d)

Figure 1: Illustration of importance sampling and kernel sparsity on a two-dimensional synthetic data set containing 1,000 points along 10 concentric circles (100 data points from each cluster). Figures (a)-(c) show all the data points represented by "o" and the sampled points represented by "*". Figure (a) shows 50 points sampled using importance sampling, and Figures (b) and (c) show 50 and 100 points selected using uniform random sampling, respectively. All the 10 clusters are represented by just 50 points sampled using importance sampling, whereas the 50 points sampled uniformly are not sufficiently representative (Cluster 4 in red has no representatives). 100 points need to be uniformly sampled to represent all the clusters. The RBF kernel matrix corresponding to this data is shown in (d). Neighboring points have the same cluster label when the kernel is defined correctly for the data set.

sampled with probability p_i, defined as

$$p_i = \frac{1}{C} \left|\left| V_C^{(i)} \right|\right|_2^2, \tag{3}$$

where $V_C^{(i)}$ is the i^{th} row of V_C. The term $\left|\left| V_C^{(i)} \right|\right|_2^2$, called the statistical leverage score for data point $\mathbf{x}_i$, is an indicator of the importance of the point. A high score indicates that the corresponding data point has a high influence in the approximation of the kernel matrix.

Statistical leverage scores have been used successfully to obtain low rank matrix approximations of large matrices, for large scale data analysis operations [12, 20]. By selecting a subset of the rows with the largest statistical leverage values, we can represent the distribution of the entire data with just this subset. In [11][Lemma 2], it was shown that a good low rank approximation of the true kernel matrix can be obtained by sampling $m = \Omega(C \log C)$ data points. By adopting importance sampling in our algorithm, we obtain a good approximation of the true kernel by sampling just a fraction of the data set. Figures 1(a)-(c) illustrate the advantage of importance sampling over uniform sampling on a two-dimensional data set containing 1,000 points from 10 clusters (100 data points from each cluster). Each true cluster is a concentric circle with varying radius, as shown in Figure 1(a). Figure 1(a) also shows 50 points sampled using importance sampling. We observe that all the 10 clusters are adequately represented by the 50 sampled points. Figure 1(b) shows 50 points uniformly sampled from the data. These points do not represent all the clusters, as the probability of uniformly sampling data points from all the clusters is low. We require at least 100 uniformly sampled points to represent all 10 clusters, as shown in Figure 1(c).

Kernel Sparsity. Our algorithm uses the p-nearest neighbors ($p > C$) of each point to construct a sparse kernel matrix. The intuition behind this is the fact that, each data point is surrounded by points belonging to the same cluster in the high dimensional feature space, provided the kernel function is appropriately selected. Figure 1 illustrates this concept on the two-dimensional concentric circles data set. The RBF kernel matrix corresponding to this data is shown in Figure 1(d). Nearby data points in terms of the kernel similarity tend to have the same cluster label. This idea has been previously applied in several local learning approaches. The local learning based clustering algorithm [21] and the local spectral clustering algorithm [6] use the nearest neighbor graphs to obtain the cluster labels for the data. However, these methods require the computation of the full $N \times N$ similarity matrices, rendering them non-scalable.

Finding the nearest neighbors of a data point from amongst r points would require the computation of $O(r)$ similarities. The randomized *kd*-trees [14] technique for approximate nearest neighbor computation involves constructing multiple *kd*-trees and searching them in parallel. While a classical *kd*-tree is built by splitting the data along the dimensions with the highest variance [22], each randomized *kd*-tree splits the data along a dimension chosen randomly from the top N_d dimensions with the highest variance. A priority queue with information about the distance of each branch to the decision boundary is used to index into the multiple trees. It takes $O(r \log r)$ time to build the trees, and $O(\log r)$ time for each query. Therefore, the time taken for nearest neighbor computation is significantly reduced, especially when a large number of queries need to be performed on the same data set. We employ randomized *kd*-trees to (i) find the nearest neighbors and build the sparse kernel matrix, and (ii) to find the closest center for each data point during clustering.

The proposed algorithm offers the following advantages over the existing techniques to reduce the running time of kernel-based clustering [2, 3, 21]:

(i) It employs importance sampling, so a significantly smaller number of samples are required to approximate the kernel matrix, when compared to the approximation methods in [2, 3], which employ uniform random sampling.

(ii) The number of kernel similarity computations performed by the proposed algorithm is $O(Np)$, where the number of neighbors $p \ll m \ll N$. Therefore, its running time and memory is lesser than that of the existing approximate kernel clustering algorithms [2, 3, 6, 21].

(iii) The clustering quality is better when compared to the existing approximate kernel clustering methods, even with a relatively small number of sampled points, because the most informative samples are used to perform clustering.

(iv) Our algorithm is online in nature, i.e. the data is clustered in batches of a user-defined size B, so it can cluster very large data sets (including data streams).

3. SPARSE KERNEL *K-MEANS*

The proposed sparse kernel k-means clustering algorithm is described in Algorithm 1. The algorithm starts with the first m data points stored in a buffer S of a fixed maximum size M ($C < m < M$). Let $\mathcal{N}(\mathbf{x}_i)$ represent the p-nearest neighbors of data point $\mathbf{x}_i$ in the RKHS[1]. We construct the

[1]The nearest neighbors are found efficiently using randomized *kd*-trees. We use the kernel function $\kappa(\cdot, \cdot)$ to define the inter-point distance function.

Algorithm 1 Approximate Sparse Kernel k-means

Input:
- $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_N\}$: the data set to be clustered
- $\kappa(\cdot,\cdot) : \Re^d \times \Re^d \mapsto \Re$: kernel function
- C: the number of clusters
- m: minimum number of points to be sampled ($m > C$)
- p: Number of neighbors for calculating the sparse kernel matrix ($p < m$)
- M: maximum number of points allowed in the sample set ($m < M$)
- B: Size of each input data batch

Output: Cluster labels for the data points
Initialize $S = \{\mathbf{x}_1 \ldots \mathbf{x}_m\}$.
Set the number of batches $N_B = (N-m)/B$ and divide the remaining points in the data set $(\mathcal{D} - S)$ into batches $\{\mathcal{D}^1, \ldots, \mathcal{D}^{N_B}\}$, where $\mathcal{D}^t = \{\mathbf{x}_1^t, \ldots, \mathbf{x}_B^t\}$.
Compute the sparse kernel matrix K^0 according to (4).
Decompose K^0 as $K^0 = V_C \Sigma_C V_C^\top$.
Cluster the data points in S by executing approximate k-means (Algorithm 2) on ${V_C \Sigma_C}^{1/2}$ to obtain their cluster labels.
for $t = 1, 2, \ldots, N_B$ **do**
  for $i = 1, 2, \ldots, B$ **do**
    Calculate the probability p_i^t using (3).
    Set $S = S \cup \{\mathbf{x}_i^t\}$ with probability p_i^t.
    If $\mathbf{x}_i^t$ was added to S in Step 11, update the eigenvalues Σ_C and eigenvectors V_C using (11), and recluster the points in S by executing the approximate k-means algorithm (Algorithm 2) on ${V_C \Sigma_C}^{1/2}$, otherwise assign $\mathbf{x}_i^t$ to cluster k^*, where $k^* = \arg\min_{k \in [C]} ||c_k(\cdot) - g_t(\cdot)||^2_{\mathcal{H}_\kappa}$, $c_k(\cdot)$ is given by (8), and $g_t(\cdot)$ is the projection of $\kappa(\mathbf{x}_t^i, \cdot)$ into the subspace spanned by the eigenvectors V_C.
    If $\mathbf{card}(S) > M$, find index $q = \arg\min_l \left|\left| V_C^{(l)} \right|\right|_2^2$ and remove data point $\mathbf{x}_q$ from S.
  end for
end for

p-nearest neighbor graph K^0 for the m data points in S, defined by

$$K^0 = [K_{ij}]_{m \times m} \text{ , where} \quad (4)$$
$$K_{ij} = \begin{cases} \kappa(\mathbf{x}_i, \mathbf{x}_j) & \text{if } \mathbf{x}_i \in \mathcal{N}(\mathbf{x}_j) \text{ and } \mathbf{x}_j \in \mathcal{N}(\mathbf{x}_i), \\ 0 & \text{otherwise.} \end{cases}$$

Assuming that the kernel function is appropriately defined, nearby points in the RKHS belong to the same cluster[2]. The remaining data is clustered in batches $\{\mathcal{D}^1, \mathcal{D}^2, \ldots\}$ of size B, where $\mathcal{D}^t = \{\mathbf{x}_1^t, \ldots, \mathbf{x}_B^t\}$. Let $K^t = V_C \Sigma_C {V_C}^\top$, where $\Sigma_C = \mathrm{diag}(\lambda_1, \ldots, \lambda_C)$ contains the top C eigenvalues of K^t (the kernel matrix at time t), and $V_C = (\mathbf{v}_1, \ldots, \mathbf{v}_C)$ contains the corresponding eigenvectors. The matrices V_C and Σ_C are updated using each point $\mathbf{x}_i^t$ from $\mathcal{D}^t$, and the kernel matrix is updated as

$$K^t = \begin{cases} \begin{bmatrix} K^{t-1} & \varphi^\top \\ \varphi & \kappa(\mathbf{x}_i^t, \mathbf{x}_i^t) \end{bmatrix} & \text{with probability } p_i^t, \\ K^{t-1} & \text{with probability } 1 - p_i^t, \end{cases} \quad (5)$$

where φ is a sparse vector defined by $\varphi = \left[\kappa\left(\mathbf{x}_i^t, \mathbf{x}_r\right)\right]^\top$, $\mathbf{x}_r \in \mathcal{N}\left(\mathbf{x}_i^t\right) \cap S$, and p_i^t is the importance sampling probability defined in (3). Data point $\mathbf{x}_i^t$ is added to S with probability p_i^t. The cluster labels for the points in S can be obtained by solving the kernel k-means problem

$$\max_{U \in \{0,1\}^{C \times s}} \mathrm{tr}(\widetilde{U} K^t \widetilde{U}^\top), \quad (6)$$

where $s = \mathbf{card}(S)$. The cluster labels for the unsampled points can be obtained by assigning them to the closest center. The running time complexity of this step is $O(s^2)$. We further reduce this complexity by constraining the cluster centers to the subspace spanning the top eigenvectors of the kernel matrix K^t, along the lines of spectral clustering[3]. We pose the clustering problem as the following optimization problem:

$$\min_{U \in \{0,1\}^{C \times s}} \max_{\{c_k(\cdot) \in \mathcal{H}_a\}_{k=1}^C} \sum_{k=1}^{C} \sum_{i=1}^{s} \frac{U_{ki}}{s} ||c_k(\cdot) - \kappa(\mathbf{x}_i, \cdot)||^2_{\mathcal{H}_\kappa}, \quad (7)$$

where $\mathcal{H}_a = \mathrm{span}\,(\mathbf{v}_1, \ldots, \mathbf{v}_C)$. The cluster centers can be expressed as linear combinations of the eigenvectors of the kernel matrix:

$$c_k(\cdot) = \sum_{i=1}^{s} \sum_{j=1}^{C} \frac{U_{ki}}{N_k} \sqrt{\lambda_j} v_{ij} = \frac{\mathbf{u}_k}{N_k} V_C {\Sigma_C}^{1/2}, \; k \in [C], \quad (8)$$

where N_k is the number of points in the k^{th} cluster and $\mathbf{u}_k = (U_{k1}, U_{k2}, \ldots, U_{ks})^\top$.

By substituting (8) in (7), we obtain the following trace maximization problem:

$$\max_{U \in \{0,1\}^{C \times s}} \mathrm{tr}(\widetilde{U} V_C \Sigma_C V_C^\top \widetilde{U}^\top). \quad (9)$$

The above problem can be solved by executing k-means on the matrix $V_C \Sigma_C^{1/2}$. The complexity of this step is $O(sC^2)$, which can again be computationally expensive for large C. We alleviate this issue by designing an approximate variant of the k-means algorithm (Algorithm 2), similar to the filtering algorithm in [24]. The most computationally expensive step in the k-means algorithm is computing the closest center for each data point, which requires $O(sC)$ distance computations. We reduce the number of distance computations by using randomized kd-trees to find the closest centers.

The proposed algorithm is dependent on three parameters: initial sample size m, maximum buffer size M, and the number of neighbors p used to build the sparse kernel matrix. The parameters m and M should be set such that the initial and final sample sets contain representatives from all the clusters. The parameter p should be set large enough to ensure that the kernel matrix remains SPSD with rank greater than C. Heuristics to set these parameters are discussed further in Section 4.

Algorithm 2 Approximate k-means

Input:
- $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_N\}$: the data points to be clustered
- C: the number of clusters

Output: Cluster labels for the data points
Randomly initialize the cluster labels $\{l_1, l_2, \ldots l_N\}, l_i \in [C]$.
Compute the cluster centers $c_k = \sum_{l_i = k} \mathbf{x}_i, k \in [C]$.
repeat
  Build randomized kd-tree index I for the C centers [14].
  for $i = 1, 2, \ldots, N$ **do**
    Find the approximate nearest center c_{k*} of data point $\mathbf{x}_i$ using the index I.
    Set $l_i = k^*$.
  end for
  Recompute the centers $\{c_1, c_2, \ldots, c_C\}$.
until convergence

[2]Selection of the kernel function is a perennial problem in machine learning. Several articles in the literature describe techniques to learn the kernel function from the data, eg. [23].

[3]Note that the eigenvalues and eigenvectors were computed while computing the sampling probabilities (3), so the eigenvectors do not need to be re-computed for clustering.

3.1 Approximation Error

The proposed sparse kernel k-means algorithm essentially approximates the eigenvectors of the true $N \times N$ kernel matrix with the singular vectors of a sparse $N \times T$ matrix, where T is the total number of points sampled from the data set using importance sampling. In the following two lemmas, we first bound the kernel approximation error due to importance sampling and sparsification, and then bound the error incurred due to the approximation (7) for clustering. The proofs of the lemmas are omitted due to space constraints[4].

LEMMA 1. *Let K be the $N \times N$ kernel matrix and let $\bar{K}$ be the $N \times T$ kernel matrix between the N points in the data set and the T sampled points. Let $Z_C = (z_1, \ldots, z_C)$ represent the top C eigenvectors of K, and $\delta \in (0,1)$ be the smallest probability such that $(\lambda_C - \lambda_{C+1}) > 3\Delta$, where*

$$\Delta < \frac{2\lambda_1}{T} \ln \frac{2}{\delta} + \gamma |K|_F \sqrt{\frac{2\ln(2/\delta)}{TN}} \text{ and } \gamma^2 = \max_{1 \le i \le T} \sum_{j=1}^{N} \kappa^2(\mathbf{x}_i, \mathbf{x}_j).$$

Assuming $\gamma = O(\sqrt{T})$ and $\kappa(\cdot,\cdot) \le 1$,

$$\max_{1 \le i \le C} |\mathbf{v}_i - \mathbf{z}_i|_2 \quad \le \quad \frac{9\Delta}{2(\lambda_C - \lambda_{C+1})}, \tag{10}$$

with probability $1 - \delta$.

The proof of this lemma involves finding the upper bound of the difference between the canonical angles of the subspaces spanned by the eigenvectors Z_C and V_C. This lemma shows that when the difference $(\lambda_C - \lambda_{C+1})$ is sufficiently large, the error between the eigenvectors of the true kernel matrix and the singular vectors of the sparse kernel matrix constructed by our algorithm is small.

In the following lemma, we show that the error incurred due to the approximation (7) is well-bounded, provided that the tail of the eigenspectrum is fast decaying, which is true for most real data sets:

LEMMA 2. *Let E and E_a represent the optimal clustering errors in (6) and (9), respectively. We have*

$$|E - E_a| \le \sum_{i=C+1}^{s} \lambda_i.$$

3.2 Complexity

The most computationally intensive operations in the proposed algorithm are: (i) computing the $m \times m$ kernel matrix K^0 (Step 5) and finding its eigenvectors to obtain the leverage scores (Step 6), and (ii) updating the eigenvectors in each iteration and clustering them using the approximate k-means algorithm (Step 12). In order to obtain the eigenvalues and eigenvectors of an $s \times s$ kernel matrix K^t (where s is the number of data points in the buffer S), we need to perform eigendecomposition of K^t. Naive implementations of eigendecomposition take $O(s^3)$ time. We can reduce this time by making two modifications to the algorithm:

(i) Use efficient algorithms such as Lanczos, and subspace iteration methods to decompose the $m \times m$ kernel matrix K^0 [26]. This reduces the running time complexity of this step to $O(mp + m)$. In our implementation, we used the `svds` function in MATLAB to obtain the top C eigenvalues and eigenvectors of K^0.

(ii) Update the eigenvectors V_C incrementally in each iteration of the algorithm using fast update mechanisms. For instance, using the rank-1 update mechanism in [27], we update the eigenvectors in $O(sp + p^3)$ time. Given $K^t = V_C \Sigma_C V_C^\top$, and vector $\varphi \in \Re^s$, this method finds the eigendecomposition of $\left(K^t + \varphi\varphi^\top\right)$ as

$$K^t + \varphi\varphi^\top \quad = \quad \left[V \quad \frac{w}{||w||}\right] \Sigma' \left[V \quad \frac{w}{||w||}\right]^\top \tag{11}$$

where $w = \left(I - VV^\top\right)\varphi$ is the component of K^t that is orthogonal to V, and Σ' contains the dominant eigenvalues of the sparse matrix

$$\begin{bmatrix} \Sigma & V^\top\varphi \\ \varphi^\top V & ||w|| \end{bmatrix}.$$

This method also eliminates the need to store the kernel matrix K^t in memory. After the matrix K^0 and its eigenvectors are obtained, only the vector φ in (5) is required to update V_C and Σ_C.

The approximate k-means algorithm first builds multiple randomized kd-trees containing the C cluster centers, and an index into these trees, which takes $O(C \log C)$ time. It then finds the approximate nearest neighbors for each data point in S in $O(s \log C)$ time, with an ϵ approximation error. Therefore, the total time for clustering s points using the approximate k-means algorithm is $O(C \log Cl + s \log Cl)$, where l is the number of iterations required for convergence. Instead of clustering the buffered data each time a point is added, we employ a *lazy reclustering* approach, by which we perform the clustering after every L data point additions. Each unsampled data point can be assigned a cluster label by finding the closest center in $O(\log C)$ time.

In summary, the overall running time complexity of the proposed sparse kernel k-means algorithm is $O(Npd + mp + m + TC \log Cl + TM \log Cl + N \log C)$, where T is the total number of points sampled from the data set. Therefore, the running time complexity can be simplified as $O(Npd + N \log C)$, assuming $\max(mp, TCl, TMl) \ll N$. This is significantly faster than the kernel k-means algorithm and the approximate kernel clustering algorithms, which have $O(N^2 d + N^2 C)$ and $O(Nmd + NmC)$ running time complexities, respectively. The amount of memory required is $O(mp + Md + MC)$, for storing the initial kernel matrix K^0, the data points in the buffer and the eigenvectors of the kernel matrix.

Table 2: Data sets used to evaluate the proposed algorithm. The number of points (N), the data dimensionality (d) and the number of clusters (C) are specified.

Data set	N	d	C
CIFAR-100 [28]	60,000	384	100
Imagenet [29]	1,262,102	900	164
Youtube	10,143,254	6,647	10,000
Tiny [30]	79,302,017	384	10,000

4. EXPERIMENTAL RESULTS

4.1 Data sets

We demonstrate the effectiveness of the proposed sparse kernel k-means algorithm on the following four data sets. These data sets were chosen to demonstrated the efficiency and effectiveness of our algorithm for large N, d and C (See Table 2):

- **Tiny and CIFAR-100 [28, 30]:** The Tiny Image data set contains $79,302,017$ unique 32×32 color images, downloaded using $75,062$ non-abstract English nouns from the Wordnet database as search queries. The CIFAR-100 image data set is a manually labeled subset of the Tiny data set containing $60,000$ images

[4]Proofs are outlined in the technical report [25].

from 100 classes. The images in the Tiny data set are represented by 384-dimensional GIST features; the CIFAR-100 images are represented using SIFT descriptors.

- **Imagenet [29]:** The Imagenet data set contains about 14 million images organized according to a conceptual "synset" hierarchy. We downloaded $1,262,102$ images from $1,000$ synsets, and merged the leaf nodes in the synset tree based on their similarity to form a 164-class data set. The images are represented using SIFT descriptors quantized into 900 bag-of-visual-words.
- **Youtube:** We used the Youtube Search API[5] to download the video title, description and the video thumbnail (which usually contains the key frame in the video) of $10,143,254$ videos using $26,000$ nouns from Wordnet as search queries. For each video, we eliminated stop words from the title and description to obtain a vocabulary containing $6,135$ terms, and extracted the corresponding tf-idf (term frequency-inverse document frequency) features. Each record was represented by a $6,647$-dimensional feature vector obtained by concatenating the tf-idf features with the GIST features of the thumbnail image.

4.2 Baselines and Parameters

We compared the performance of the proposed algorithm with the kernel k-means [5] algorithm on the CIFAR-100 data set. It is infeasible to execute the kernel k-means algorithm on the remaining three data sets. We also evaluated its performance against the k-means algorithm. Finally, we compared our algorithm with the approximate kernel k-means algorithm [2], where the data is sampled with uniform probability, to study the effect of importance sampling.

We used the universal RBF kernel for the proposed algorithm and the kernel-based baseline algorithms (kernel k-means and approximate kernel k-means) on the Tiny and Imagenet data sets, as it can be computed efficiently. For the CIFAR-100 data set, we employed the spatial pyramid kernel to show that our algorithm is applicable to a variety of kernels. For the Youtube data set, which contains both text and image features, we used a combination of the cosine similarity and the RBF kernel, defined as

$$\kappa(\mathbf{x}_a, \mathbf{x}_b) = \frac{1}{2}\left[\exp\left(-\lambda\|g_a - g_b\|^2\right) + \frac{f_a^\top f_b}{\|f_a\|\|f_b\|}\right],$$

where f_a and g_a denote the tf-idf and GIST features for data point $\mathbf{x}_a$, respectively. We tuned the kernel width for the RBF kernel using grid search in the range $[0, 1]$ to obtain the best performance for the proposed and the baseline algorithms.

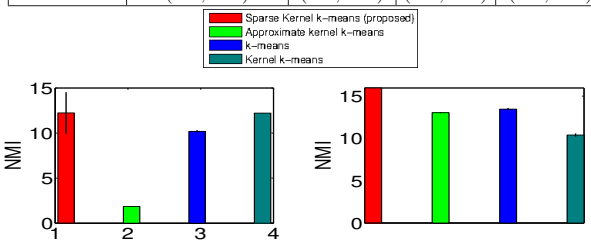
We varied the initial sample set size from $m = 5,000$ to $m = 20,000$, and the number of neighbors from $p = 1,000$ to m in multiples of $5,000$. The maximum sample set size was set to $M = 50,000$. The number of clusters C was set equal to the true number of classes in the data set for the CIFAR-100 and Imagenet data sets. The true number of classes is unknown for the Youtube and Tiny data sets, so we set the number of clusters equal to $10,000$. The batch size B was set equal to the initial sample size m.

We implemented all the algorithms in MATLAB and executed them 10 times each on a 2.8 GHz processor. Different permutations of the data set were input to the clustering algorithms in each run. We present the results (mean and variance) over the 10 runs. The memory used was constrained to 60GB. We used the randomized kd-trees implementation in the FLANN library [14] to find the approximate nearest neighbors in the proposed algorithm. The distance function used by the library was defined as the inverse of the kernel similarity function. The kd-tree parameters were set as: the number of dimensions N_d to 5, the number of trees to 8 and the approximation error to $\epsilon = 1e^{-16}$.

[5] `https://developers.google.com/youtube/v3/`

Table 3: Running time of the proposed and baseline algorithms in seconds. *It is not feasible to execute kernel k-means on the Imagenet, Youtube and Tiny data sets due to their large size. The running time of kernel k-means on these data sets is obtained by first finding the cluster centers using a randomly chosen subset of $50,000$ data points, and then assigning the remaining points to the closest cluster center.

Data set	Sparse Kernel k-means (proposed)	Approx. kernel k-means	k-means	Kernel k-means
CIFAR-100	49,887 ($\pm$93)	11,394 ($\pm$600)	1,507 ($\pm$332)	117,513 ($\pm$211)
Imagenet	74,794 ($\pm$870)	16,023 ($\pm 3,577$)	240,722 ($\pm 5,351$)	182,311* ($\pm 14,916$)
Youtube	217,533 ($\pm 1,264$)	57,096 ($\pm 2,196$)	145,039 ($\pm 1,436$)	679,061* ($\pm 2,284$)
Tiny	343,560 ($\pm 2,528$)	371,004 ($\pm 1,588$)	359,291 ($\pm 7,045$)	704,656* ($\pm 8,482$)



(a) CIFAR-100 (b) Imagenet

Figure 2: NMI (%) of algorithms on the CIFAR-100 and the Imagenet data sets. The NMI of kernel k-means on the Imagenet data set was obtained by executing the algorithm on a randomly chosen subset of $50,000$ data points to find the cluster centers, and assigning the remaining points to the closest cluster center.

4.3 Experimental Results

Clustering efficiency and accuracy: Table 3 compares the running time (mean and variance) of our algorithm with the approximate kernel k-means, kernel k-means and k-means algorithms, when the parameters $m = 20,000$ and $p = 1,000$. On the CIFAR-100 data set, the proposed algorithm took longer than the k-means algorithm, as expected, because of the additional time required for kernel computation and eigensystem calculation. It also took longer than the approximate kernel k-means algorithm, as it performs importance sampling by calculating and updating the eigenvectors of the sparse kernel matrix. On the other hand, the approximate kernel k-means algorithm selects the subset of the data using uniform random sampling. The proposed algorithm, the approximate kernel k-means, and the k-means algorithms were significantly faster than the kernel k-means algorithm. Our algorithm spent more time in updating the eigenvectors and finding the leverage scores, than clustering the eigenvectors to obtain the cluster labels. Similar performance was observed on the Imagenet, Youtube and Tiny data sets. The proposed algorithm was also faster than k-means on the Imagenet data set, because k-means took longer to converge. It is infeasible to compute the full kernel matrix for the Imagenet, Youtube and Tiny data sets, so we were unable to execute kernel k-means on them. For these data sets, we first executed kernel k-means on a $50,000$-sized randomly selected subset of the data, and then assigned the remaining points to the closest cluster centers. The proposed algorithm was also faster than this implementation of kernel k-means. Our algorithm was also more accurate than

Table 5: Effect of the size of the neighborhood p on the running time and NMI of the sparse kernel k-means algorithm.

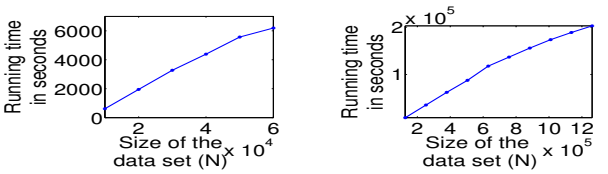
p	Running time		NMI	
	CIFAR-100	Imagenet	CIFAR-100	Imagenet
1,000	49,887 (±93)	74,794 (±870)	12.23 (±2.3)	16.15 (±0.004)
5,000	52,073 (±483)	82,880 (±21,360)	12.09 (±0.02)	17.58 (±0.10)
10,000	54,205 (±874)	192,725 (±3,874)	13.86 (±0.07)	18.01 (±0.07)
15,000	55,062 (±837)	247,911 (±7,789)	14.00 (±0.01)	18.23 (±0.004)

this kernel k-means implementation on the Imagenet data set.

We analyze the accuracy of the proposed sparse kernel k-means using the CIFAR-100 and Imagenet data sets. As the true class labels for the Youtube and Tiny data sets are not available, we were unable to quantify the clustering quality on these data sets. Figure 2 shows the Normalized Mutual Information (NMI) values with respect to the true class labels, for each of the algorithms on the CIFAR-100 and Imagenet data sets. In Figure 2(a), it is observed that the NMI achieved by our algorithm is close to that of the kernel k-means algorithm. It outperformed both k-means and approximate kernel k-means, due to the fact that it samples the most informative points from the data sets.

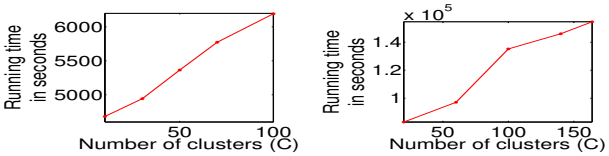
Parameter Sensitivity: Our algorithm relies on three parameters: the initial and maximum sample sizes m and M, and the size of the neighborhood p. We evaluated the effect of each of these parameters using the CIFAR-100 and Imagenet data sets.

- **Initial sample size m:** The initial sample used to construct the kernel K^0 and obtain the initial cluster labels plays a crucial role in the performance of our algorithm, as shown in Table 4. As expected, the running time of both the proposed and the approximate kernel k-means algorithms increased as the initial sample size increased from $m = 5,000$ to $m = 20,000$. As m increased, the size of the initial kernel K^0, and the time to compute and decompose it into its eigenvalues and eigenvectors increased proportionately. The initial sample also determines the number of points sampled from the data as each input batch is processed. More data points were sampled and added to the buffer S, if the initial sample did not contain sufficient number of representative points. For instance, on the CIFAR-100 data set, the number of points added to S decreased from 453 to 69 as m increased from $5,000$ to $20,000$. The time to cluster increased as more points were added to the buffer. The NMI values achieved by our algorithm increased considerably as the sample size m increased. Even with just $5,000$ data points in the initial sample, our algorithm was able to achieve about 9.5% NMI. On the other hand, the approximate kernel k-means algorithm was unable to achieve the same with even $20,000$ samples. The best performance was obtained when $m > C \log C$.
- **Maximum sample size M:** The parameter M controls the buffer size. In our experiments, we set $M = 50,000$. We found that this parameter is not as critical as m, provided that it is set large enough to accommodate for a sufficiently representative sample set. If M is small, more time is spent in removing the points from the buffer, to accommodate the new points.
- **Size of the neighborhood p:** Table 5 shows how the running time and the NMI values on the CIFAR-100 and Imagenet data sets are affected, as the number of neighbors p used to construct the sparse kernel similarity increased from $1,000$ to $15,000$, while m was fixed at $20,000$. The running time increased significantly from $p = 1,000$ to $p = 15,000$ on both the data sets, as a larger number of similarity computations needed to be performed as the value of p increased. Although increasing p slightly increases the NMI values, the increase in the running time does not compensate for the marginal increase in NMI.



(a) CIFAR-100 (b) Imagenet

Figure 3: Scalability of the proposed algorithm with respect to the size of data, N.



(a) CIFAR-100 (b) Imagenet

Figure 4: Scalability of the proposed algorithm with respect to the number of clusters, C.

Scalability: To examine the scalability with respect to the size of the data set N, we varied the number of data points input to the algorithm from the CIFAR-100 and Imagenet data sets from $10,000$ to the true size of the data sets ($60,000$ and $1,262,102$ for CIFAR-100 and Imagenet, respectively). Similarly, to test the scalability with respect to the number of clusters C, we varied C from 10 to the true number of classes in the data sets (100 and 164 for the CIFAR-100 and Imagenet, respectively). The parameters m and p were set to $5,000$ and $1,000$ respectively. Figures 3 and 4 show that the proposed algorithm is linearly scalable with respect to the size of the data set, and nearly logarithmically scalable with respect to the number of clusters, in accordance with the complexity analysis in Section 3.2.

5. CONCLUSIONS

We have proposed a clustering algorithm called the sparse kernel k-means algorithm to efficiently and effectively cluster data sets with large number of points (N), dimensionality (d), and number of clusters (C). By sampling the data points based on their novelty, defined in terms of the statistical leverage scores, we only store a very small number of the most informative points in the data. We need to compute the kernel similarity of the data points only with respect to these sampled points. This sampling strategy effectively reduces the run time complexity and memory requirements. We further reduce the running time complexity by introducing sparsity into the kernel, based on the assumption that nearby points in the kernel space have similar cluster labels. We demonstrated that the proposed algorithm is scalable and accurate using several large benchmark data sets containing millions of points, hundreds of features and up to $10,000$ clusters. By utilizing a parallel scheme for updating the eigensystem of the sparse kernel matrix and finding the statistical leverage scores, the proposed algorithm can be easily parallelized.

6. REFERENCES

[1] Big Data in 2020., Dec 2014. IDC and EMC Corp. Report.

Table 4: Comparison of the running time (in seconds) and NMI (%) of the proposed sparse kernel k-means algorithm and the approximate kernel k-means algorithm on the CIFAR-100 and the Imagenet data sets. m represents the initial sample set size for the proposed algorithm and the subset size for the approximate kernel k-means algorithm. Approximate kernel k-means is infeasible for the Imagenet data set when $m > 10,000$ due to its large size.

m	**Running time**				**NMI**			
	CIFAR-100		**Imagenet**		**CIFAR-100**		**Imagenet**	
	Sparse kernel k-means	**Approx. kernel k-means**	**Sparse kernel k-means**	**Approx. kernel k-means**	**Sparse kernel k-means**	**Approx. kernel k-means**	**Sparse kernel k-means**	**Approx. kernel k-means**
5,000	6,192 (±424)	1,693 (±339)	24,029 (±4,469)	15,691 (±3,786)	9.54 (±0.01)	1.86 (±0.001)	16.31 (±0.07)	13.01 (±0.001)
10,000	18,256 (±21)	4,134 (±549)	36,669 (±603)	16,023 (±3,577)	9.76 (±0.05)	1.87 (±0.002)	15.96 (±0.10)	13.04 (±0.008)
15,000	34,192 (±2,652)	7,856 (±929)	53,142 (±3,058)	-	10.40 (±0.77)	1.87 (±0.02)	15.62 (±0.31)	-
20,000	49,887 (±93)	11,394 (±600)	74,794 (±870)	-	12.23 (±2.30)	1.84 (±0.02)	16.15 (±0.004)	-

[2] R. Chitta, R. Jin, T. C. Havens, and A. K. Jain. Approximate kernel k-means: Solution to large scale kernel clustering. In *Proc. of the International Conf. on Knowledge Discovery and Data Mining*, pages 895–903, 2011.

[3] C. Fowlkes, S. Belongie, F. Chung, and J. Malik. Spectral grouping using the Nystrom method. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, pages 214–225, 2004.

[4] T. Liu, C. Rosenberg, and H. A. Rowley. Clustering billions of images with large scale nearest neighbor search. In *Proc. of the IEEE Workshop on Applications of Computer Vision*, pages 28–33, 2007.

[5] M. Girolami. Mercer kernel-based clustering in feature space. *IEEE Trans. on Neural Networks*, 13(3):780–784, 2002.

[6] U. Von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[7] A. Ben-Hur, D. Horn, H. T. Siegelmann, and V. Vapnik. Support vector clustering. *Journal of Machine Learning Research*, 2:125–137, 2002.

[8] L. Xu, J. Neufeld, B. Larson, and D. Schuurmans. Maximum margin clustering. In *Proc. of the Conf. on Neural Information Processing Systems*, pages 1537–1544, 2004.

[9] I. S. Dhillon, Y. Guan, and B. Kulis. Kernel k-means: Spectral clustering and normalized cuts. In *Proc. of the International Conf. on Knowledge Discovery and Data Mining*, pages 551–556, 2004.

[10] H. Zha, X. He, C. Ding, M. Gu, and H. D. Simon. Spectral relaxation for k-means clustering. In *Proc. of the Conf. on Neural Information Processing Systems*, pages 1057–1064, 2001.

[11] A. Gittens and M. W. Mahoney. Revisiting the nystrom method for improved large-scale machine learning. *arXiv preprint arXiv:1303.1849*, 2013.

[12] S. Chatterjee and A. S. Hadi. Influential observations, high leverage points, and outliers in linear regression. *Statistical Science*, 1(3):379–393, 1986.

[13] P. Drineas, M. Magdon-Ismail, M. W. Mahoney, and D. P. Woodruff. Fast approximation of matrix coherence and statistical leverage. *Journal of Machine Learning Research*, 13(1):3475–3506, 2012.

[14] M. Muja and D. G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240, 2014.

[15] A. K. Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666, 2010.

[16] M. Ester, H. P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of the International Conf. on Knowledge Discovery and Data Mining*, pages 226–231, 1996.

[17] P. A. Traganitis, K. Slavakis, and G. B. Giannakis. Sketch and validate for big data clustering. *arXiv preprint arXiv:1501.05590*, 2015.

[18] A. Elgohary, A. K. Farahat, M. S. Kamel, and F. Karray. Embed and conquer: Scalable embeddings for kernel k-means on mapreduce. *CoRR abs/1311.2334*, 2013.

[19] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[20] C. Boutsidis, A. Zouzias, M. W. Mahoney, and P. Drineas. Randomized dimensionality reduction for k-means clustering. *IEEE Trans. on Information Theory*, 61(2):1045, 2015.

[21] M. Wu and B. Schölkopf. A local learning approach for clustering. In *Proc. of the Conf. on Neural Information Processing Systems*, pages 1529–1536, 2006.

[22] A. W. Moore. An introductory tutorial on kd-trees. Technical report, Department of Computer Science, Carnegie Mellon University, 1991.

[23] B. Liu, S. Xia, and Y. Zhou. Unsupervised non-parametric kernel learning algorithm. *Knowledge-Based Systems*, 44:1–9, 2013.

[24] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 24(7):881–892, 2002.

[25] R. Chitta, A. K. Jain, and R. Jin. Sparse kernel clustering of massive high dimensional data sets with large number of clusters. Technical Report MSU-CSE-15-10, Computer Science and Engineering, Michigan State University, 2015.

[26] M. W. Berry. Large-scale sparse singular value computations. *International Journal of Supercomputer Applications*, 6(1):13–49, 1992.

[27] M. Brand. Fast low-rank modifications of the thin singular value decomposition. *Linear Algebra and its Applications*, 415(1):20–30, 2006.

[28] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, Department of Computer Science, University of Toronto, 2009.

[29] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[30] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 30(11):1958 –1970, 2008.