KERNEL-BASED CLUSTERING OF BIG DATA

By

Radha Chitta

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Computer Science – Doctor of Philosophy

2015

ABSTRACT

KERNEL-BASED CLUSTERING OF BIG DATA

By

Radha Chitta

There has been a rapid increase in the volume of digital data over the recent years. A study by IDC and EMC Corporation predicted the creation of 44 zettabytes (10^{21} bytes) of digital data by the year 2020. Analysis of this massive amounts of data, popularly known as *big data*, necessitates highly scalable data analysis techniques. Clustering is an exploratory data analysis tool used to discover the underlying groups in the data. The state-of-the-art algorithms for clustering big data sets are *linear* clustering algorithms, which assume that the data is linearly separable in the input space, and use measures such as the Euclidean distance to define the inter-point similarities. Though efficient, linear clustering algorithms do not achieve high cluster quality on real-world data sets, which are not linearly separable. Kernel-based clustering algorithms employ non-linear similarity measures to define the inter-point similarities. As a result, they are able to identify clusters of arbitrary shapes and densities. However, kernel-based clustering techniques suffer from two major limitations:

(i) Their running time and memory complexity increase quadratically with the increase in the size of the data set. They cannot scale up to data sets containing billions of data points.

(ii) The performance of the kernel-based clustering algorithms is highly sensitive to the choice of the kernel similarity function. Ad hoc approaches, relying on prior domain knowledge, are currently employed to choose the kernel function, and it is difficult to determine the appropriate kernel similarity function for the given data set.

In this thesis, we develop scalable approximate kernel-based clustering algorithms using random sampling and matrix approximation techniques. They can cluster big data sets containing billions

of high-dimensional points not only as efficiently as linear clustering algorithms but also as accurately as classical kernel-based clustering algorithms.

Our first contribution is based on the premise that the similarity matrices corresponding to big data sets can usually be well-approximated by low-rank matrices built from a subset of the data. We develop an approximate kernel-based clustering algorithm, which uses a low-rank approximate kernel matrix, constructed from a uniformly sampled small subset of the data, to perform clustering. We show that the proposed algorithm has linear running time complexity and low memory requirements, and also achieves high cluster quality, when provided with sufficient number of data samples. We also demonstrate that the proposed algorithm can be easily parallelized to handle distributed data sets. We then employ non-linear random feature maps to approximate the kernel similarity function, and design clustering algorithms which enhance the efficiency of kernel-based clustering, as well as label assignment for previously unseen data points.

Our next contribution is an online kernel-based clustering algorithm that can cluster potentially unbounded stream data in real-time. It intelligently samples the data stream and finds the cluster labels using these sampled points. The proposed scheme is more effective than the current kernel-based and linear stream clustering techniques, both in terms of efficiency and cluster quality.

We finally address the issues of high dimensionality and scalability to data sets containing a large number of clusters. Under the assumption that the kernel matrix is sparse when the number of clusters is large, we modify the above online kernel-based clustering scheme to perform clustering in a low-dimensional space spanned by the top eigenvectors of the sparse kernel matrix. The combination of sampling and sparsity further reduces the running time and memory complexity.

The proposed clustering algorithms can be applied in a number of real-world applications. We demonstrate the efficacy of our algorithms using several large benchmark text and image data sets. For instance, the proposed batch kernel clustering algorithms were used to cluster large image data sets (e.g. Tiny) containing up to 80 million images. The proposed stream kernel clustering algorithm was used to cluster over a billion tweets from Twitter, for hashtag recommendation.

To My Family

ACKNOWLEDGMENTS

"Life is a continuous learning process.
Each day presents an opportunity for learning." - Lailah Gifty Akita, Think Great: Be Great

Every day during my PhD studies has been a great opportunity for learning, thanks to my advisors, colleagues, friends, and family. I am very grateful to my thesis advisor Prof. Anil K. Jain, who has been a wonderful mentor. His ability to identify good research problems has always been my inspiration. I am motivated by his energy, discipline, meticulousness and passion for research. He has taught me to plan and prioritize my work, and present it in a convincing manner. I am also very thankful to Prof. Rong Jin, with whom I had the privilege of working closely. Under his guidance, I have learnt how to formalize a problem, and develop coherent solutions to the problem, using different machine learning tools. I am inspired by his extensive knowledge and hard-working nature.

I would like to thank my PhD committee members, Prof. Pang-Ning Tan, Prof. Shantanu Chakrabartty, and Prof. Selin Aviyente for their valuable comments and suggestions. Prof. Pang-Ning Tan was always available when I needed help, and provided very useful suggestions.

I am grateful to several other researchers who have mentored me at various stages of my research. I have had the privilege of working with Dr. Suvrit Sra and Dr. Francesco Dinuzzo, at the Max Planck Institute for Intelligent Systems, Germany. I would like to thank them for giving me an insight into several emerging problems in machine learning. I thank Dr. Ganesh Ramesh from Edmodo for providing me the opportunity to learn more about natural language processing, and building scalable solutions. Dr. Timothy Havens was very helpful when we were working together during the first year of my PhD.

I would like to thank my lab mates and friends: Shalini, Soweon, Serhat, Zheyun, Jinfeng,

Mehrdad, Kien, Alessandra, Abhishek, Brendan, Jung-Eun, Sunpreet, Inci, Scott, Lacey, Charles, and Keyur. They made my life at MSU very memorable. I would like to specially thank Serhat for all the helpful discussions, and Soweon for her support and encouragement. I am thankful to Linda Moore, Cathy Davison, Norma Teague, Katie Trinklein, Courtney Kosloski and Debbie Kruch for their administrative support. Many thanks to the CSE and HPCC administrators, specially Kelly Climer, Adam Pitcher, Dr. Dirk Colbry, and Dr. Benjamin Ong.

Last but not the least, I would like to thank my family. I am deeply indebted to my husband Praveen, without whose support and motivation, I would not have been able to pursue and complete my PhD. My parents, my sister and my parents-in-law have been very supportive throughout the past five years. I was inspired by my father Ramamurthy to pursue higher studies, and strive to make him proud. I would like to specially mention my mother Sudha Lakshmi, who has been my role model and inspiration. I can always count on her to encourage me and uplift my spirits.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

List Of Algorithms

Chapter 1

Introduction

Over the past couple of decades, great advancements have been made in data generation, collection and storage technologies. This has resulted in a *digital data explosion*. Data is uploaded everyday by billions of users to the web in the form of text, image, audio and video, through various media such as blogs, e-mails, social networks, photo and video hosting services. It is estimated that 204 million e-mail messages are exchanged every minute[1]; over a billion users on Facebook share 4.75 billion pieces of content every half hour, including 350 million photos and 4 million videos[2]; and 300 hours of videos are uploaded to YouTube every minute[3]. In addition, a large amount of data about the web users and their web activity is collected by a host of companies like Google, Microsoft, Facebook and Twitter. This data is now popularly termed as *Big Data* [105].

Big data is formally defined as "high volume, high velocity, and/or high variety information assets that require new forms of processing to enable enhanced decision making, insight discovery and process optimization". It is characterized by the 3V's - Volume, Velocity, and Variety. Volume indicates the scale of the data. A study by IDC and EMC Corporation predicted the creation of 44 zettabytes (10^{21} bytes) of digital data by the year 2020 (See Figure 1.1) [2]. This boils

[1] http://mashable.com/2014/04/23/data-online-every-minute

[2] http://www.digitaltrends.com/social-media /according-to-facebook-there-are-350-million-photos-uploaded -on-the-social- network-daily-and-thats-just-crazy

[3] https://www.youtube.com/yt/press/statistics.html

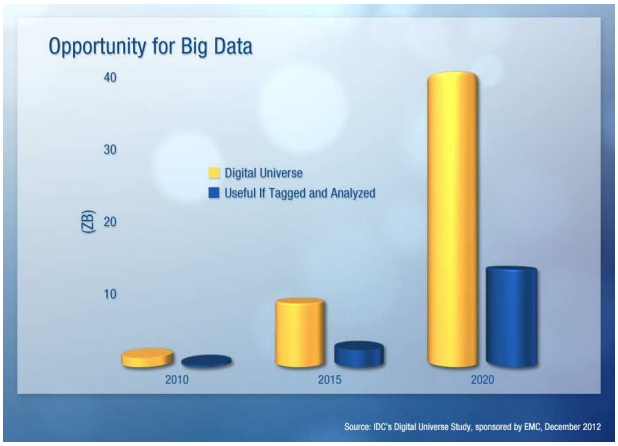


Figure 1.1 Emerging size of the digital world. Image from [2].

down to about 2.3 zettabytes of data generated every day. Velocity relates to real-time processing of streaming data in applications like computer networks and stock exchanges. The New York Stock Exchange captures about 1 TB of trade information during each trading session. Real-time processing of this data can aid a trader in making important trade decisions. Variety pertains to the heterogeneity of the digital data. Both structured data such as census records and legal records, and unstructured data like text, images and videos from the web form part of big data. Specialized techniques may be needed to handle different formats of the data. Other attributes such as reliability, volatility and usefulness of the data have been added to the definition of big data over the years. Virtually every large business is interested in gathering large amounts of data from its customers and mining it to extract useful information in a timely manner. This information helps the business provide better service to its customers and increase its profitability.

About 23% of this humongous amount of digital data is believed to contain useful information that can be leveraged by companies, government agencies and individual users[4]. For instance, a partial "blueprint" of every user on the web can be created by combining the information from their Facebook/Google profiles, status updates, Twitter tweets, metadata of their photo and video uploads, webpage visits, and all sorts of other minute data. This gives an insight into the interests

[4]http://www.mckinsey.com/insights/business_technology/big_data_the_next_frontier_for_innovation

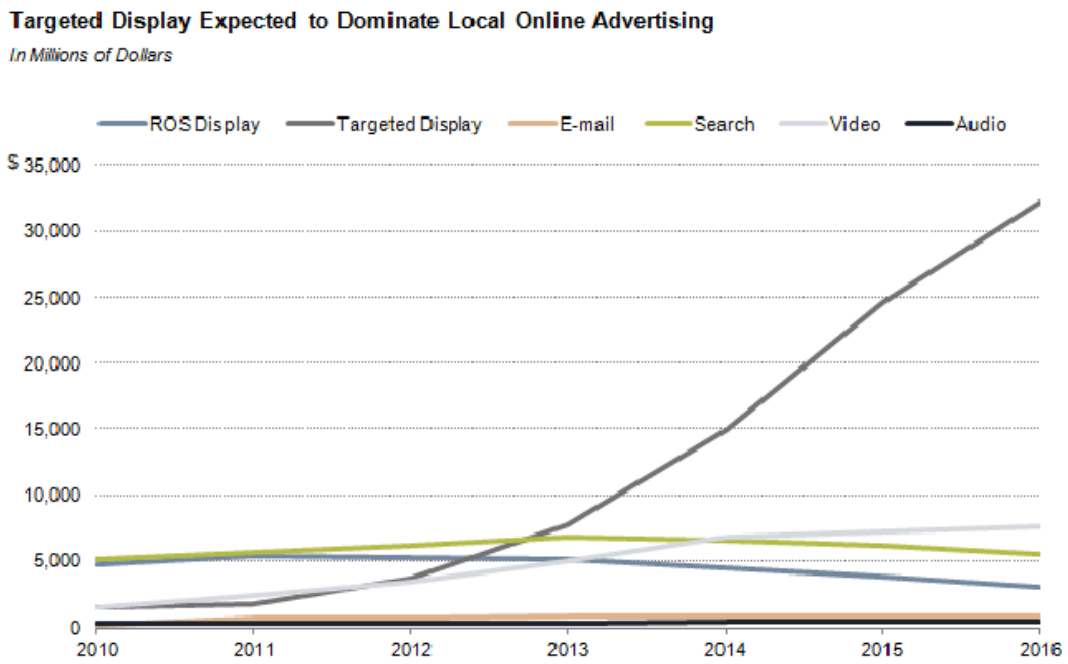


Figure 1.2 Growth of Targeted Display Advertising. Image from [59].

and needs of the users, thereby allowing companies to target a select group of users for their products. Users prefer online advertisements that match their interests over random advertisements. Figure 1.2 shows the tremendous growth that has been achieved in targeted advertising over the years, as a consequence of using data analytics[5] to understand the behavior of web users [59].

Big Data analytics has also lead to the development of new applications and services like Microsoft's HealthVault[6], a platform that enables patients to compile personal health information from multiple sources into a single online repository, and coordinate their health management with other users. Applications such as Google Flu Trends[7] and Dengue Trends[8] predicted the disease outbreak well before the official CDC (US Centers for Disease Control and Prevention)

[5]Data analytics is the science of examining data with the purpose of inferring useful information, and making decisions and predictions based on the inferences. It encompasses a myriad of methodologies and tools to perform automated analysis of data [1].

[6]https://www.healthvault.com/us/en/overview

[7]http://www.google.org/flutrends

[8]http://www.google.org/denguetrends

and EISS (European Influenza Surveillance Scheme) reports are published, based on aggregated search activity, reducing the number of people affected by the disease [71].

1.1 Data Analysis

Data analysis is generally divided into exploratory and confirmatory data analysis [174]. The purpose of exploratory analysis is to discover patterns and model the data. Exploratory data analysis is usually followed by a phase of confirmatory data analysis which aims at model validation. Several statistical methods have been proposed to perform data analysis. Statistical pattern recognition and machine learning is concerned with predictive analysis, which involves discovering relationships between objects and predicting future events, based on the knowledge obtained. Pattern recognition comprises of three phases: data representation, learning and inference.

1.1.1 Data Representation

Data representation involves selecting a set of features to denote the objects in the data set. A d-dimensional vector $\mathbf{x} = (x_1, \ldots, x_d)^\top$ denotes each object, where $x_p,\ p \in [d]$ represents a feature. The features may be numerical, categorical or ordinal. For instance, a document may be represented using the words in the document; in which case each x_p denotes a word in the document. An image may be represented using the pixel intensity values. In this case, x_p is the numerical intensity value at the p^{th} pixel. The representation employed dictates the kind of analysis the can be performed on the data set, and the interpretation of the results of analysis. Therefore, it is important to select the correct representation. In most applications, prior domain knowledge is useful in selecting the object representation. Recently, deep learning techniques have been employed to automatically learn the representation for objects [20].

1.1.2 Learning

After a suitable representation is chosen, the data is input to a learning algorithm which fits a model to the data.

The simplest learning task is that of **supervised learning**, also termed as classification [97]. The goal of supervised learning is to derive a function that maps the set of input objects to a set of targets (classes), using *labeled* training data. For instance, given a set of tagged images, the learner analyzes the images and learns a function mapping the images to their tags. Supervised learning finds use in many applications such as object recognition, spam detection, intrusion detection, and machine translation.

Unfortunately, only about 3% of the potentially useful data on the web is labeled (e.g. tags for objects in images), and it is extremely expensive to obtain the labels for the massive amount of data, making supervised learning difficult in most big data applications [2]. Of late, crowdsourcing tools such as Amazon Mechanical Turk[9] have been used to obtain labels for the data items, from multiple users over the web [29]. However, labels obtained through such approaches can be unreliable and ambiguous. For example, in the task of image tagging through crowdsourcing, one user may tag the image of a poodle with the label "dog", whereas another user may label it as "animal" (i.e. usage of hypernyms versus hyponyms). The same tag "jaguar" could apply to both the car as well as the animal (polysemy). Spammers can intentionally generate wrong labels leading to noise in the data. Additional efforts are needed to handle these issues [138, 185].

Semi-supervised learning techniques alleviate the need for labeling large data sets by utilizing a large pool of unlabeled objects in conjunction with a relatively small set of labeled objects to learn a classifier [189]. It has been found that the classifiers learnt through semi-supervised learning methods can be more accurate than those learnt using labeled data alone, because the unlabeled data allows the learner to explore the underlying structure of the data. Though semi-supervised learning methods mitigate the labeling problem associated with supervised learning methods to

[9] https://www.mturk.com/mturk

some extent, they are still susceptible to same issues as the supervised learning techniques. Moreover, it is expensive to obtain supervision in applications such as stock market analysis, where high level of expertise is required to identify the stock trends [130].

Unsupervised learning tasks involve finding the hidden structure in data. Unlike supervised and semi-supervised learning, these tasks do not require the data to be labeled, thereby avoiding the cost of tagging the data and allowing one to leverage the abundant data corpus. Examples of unsupervised learning tasks include density estimation, dimensionality reduction, feature selection and extraction, and clustering [83].

Clustering, also known as unsupervised classification, is one of the primary approaches to unsupervised learning. The purpose of clustering is to discover the natural grouping of the input objects. One of the goals of clustering is to summarize and compress the data, leading to efficient organization and convenient access of the data. It is often employed as a precursor to classification. The data is first compressed using clustering, and a supervised learning model is built using only the compressed data. For instance, in the image tagging problem, if the learner was only provided with a large number of untagged images, the images can be grouped into clusters based on a predefined similarity. Each cluster can be represented by a small set of prototype images, and the labels for these representative images obtained through crowdsourcing, which can then be used to learn a tagging function in a supervised manner. This process is cheaper and more reliable than obtaining the labels for all the images. Clustering finds use in a multitude of applications such as web search, social network analysis, image retrieval, gene expression analysis, market analysis and recommendation systems [90].

1.1.3 Inference

In this phase, the learnt model is used for decision making and prediction, as required by the application. For example, in the image tagging problem, the model comprising the mapping function can be used to predict the tags corresponding to an image the learner has not seen previously. In

Table 1.1 Notation.

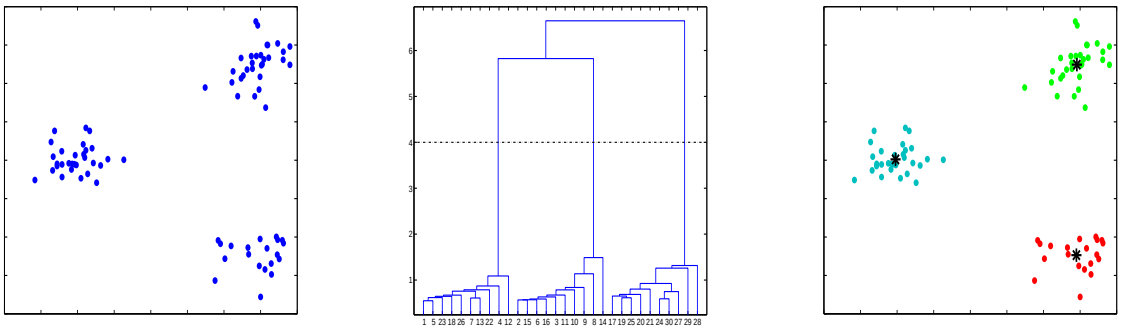
Symbol	Description
$\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$	Input data set to be clustered
$\mathbf{x}_i$	i^{th} data point
χ	Input space
$\mathcal{H}_\kappa$	Feature space/ Reproducing Kernel Hilbert Space (RKHS)
$\|\cdot\|_{\mathcal{H}_\kappa}$	Functional norm in RKHS
d	Dimensionality of the input space
n	Number of points in the data set
C	Number of clusters
$U = (\mathbf{u}_1, \ldots, \mathbf{u}_C)^\top$	Cluster membership matrix $(C \times n)$
$\mathcal{P} = \{U \in \{0,1\}^{C \times n} : U^\top \mathbf{1} = \mathbf{1}\}$	Set of valid cluster membership matrices
$\mathcal{C}_k$	k^{th} cluster
$\mathbf{c}_k$	k^{th} cluster center
n_k	Number of points in the k^{th} cluster
φ	Mapping function from χ to $\mathcal{H}_\kappa$
$\kappa(\cdot, \cdot)$	Kernel function
K	Kernel matrix $(n \times n)$

social networks, clustering is employed to group users based on their gender, occupation, web activity, and other attributes, to automatically find user communities [128]. Based on the communities identified, recommendations for new connections and content can be made to the users.

In this thesis, we focus on the clustering problem. Notations used throughout this thesis are summarized in Table 1.1.

1.2 Clustering

Clustering, one of the primary approaches to unsupervised learning, is the task of grouping a set of objects into clusters based on some user-defined similarity. Given a set of n objects represented by $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, where each point $\mathbf{x}_i \in \chi$ and $\chi \subseteq \Re^d$, the objective of clustering, in most applications, is to group the points into C clusters, represented by $\{\mathcal{C}_1, \ldots, \mathcal{C}_C\}$, such that the clusters reflect the natural grouping of the objects. The definition of natural grouping is subjective,



(a) (b) (c)

Figure 1.3 A two-dimensional example to demonstrate hierarchical and partitional clustering techniques. Figure (a) shows a set of points in two-dimensional space, containing three clusters. Hierarchical clustering generates a dendrogram for the data. Figure (b) shows a dendrogram generated using the complete-link agglomerative hierarchical clustering algorithm. The horizontal axis represents the data points and the vertical axis represents the distance between the clusters when they first merge. By applying a threshold on the distance at 4 units (shown by the black dotted line), we can obtain the three clusters. Partitional clustering directly finds the C clusters in the data set. Figure (c) shows the three clusters, represented by the blue, green and red points, obtained using the *k*-means algorithm. The starred points in black represent the cluster centers.

and dependent on a number of factors including the objects in the data set, their representation, and the goal of cluster analysis. The most common objective is to group the points such that the similarity between the points within the same cluster is greater than the similarity between the points in different clusters. The structure of the clusters obtained is determined by the definition of the similarity. It is usually defined in terms of a distance function $\mathbf{d} : \chi \times \chi \rightarrow \Re$.

1.2.1 Clustering Algorithms

Historically, two type of clustering algorithms have been developed: hierarchical and partitional [88].

- Hierarchical clustering algorithms, as the name suggests, build a hierarchy of clusters; the root of the tree contains all the n points in the data set, and the leaves contain the individual points. Agglomerative hierarchical clustering algorithms start with n clusters, each with one

point, and recursively merge the clusters which are most similar to each other. Divisive hierarchical clustering algorithms, on the other hand, start with the root containing all the data points, and recursively split the data into clusters in a top-down manner. The most well-known hierarchical clustering algorithms are the single-link, complete-link and Ward's algorithms [88]. The single-link algorithm defines the similarity between two clusters as the similarity between their most similar members, whereas the complete-link algorithm defines the similarity between two clusters as the similarity of their most dissimilar members. The Ward's clustering algorithm recursively merges the clusters that leads to the least possible increase in the intra-cluster variance after merging. Figure 1.3(b) shows the complete-link dendrogram corresponding to the clusters in the two-dimensional data set in Figure 1.3(a).

- Partitional clustering algorithms, directly partition the data into C clusters, as shown in Figure 1.3(c). Popular partitional clustering algorithms include centroid-based (*k*-means, *k*-medoids) [87, 94], model-based (Mixture models, Latent Dirichlet Allocation) [24], graph-theoretic (Minimum Spanning Trees, Normalized-cut, Spectral clustering) [77, 161], and density and grid-based (DBSCAN, OPTICS, CLIQUE) algorithms [61].

From a statistical viewpoint, clustering techniques can also be categorized as parametric and non-parametric [127]. Parametric approaches to clustering assume that the data is drawn from a density $p(\mathbf{x})$ which is a mixture of parametric densities, and the goal of clustering is to identify the component densities. The centroid-based and model-based clustering algorithms fall in this category. Non-parametric approaches are based on the premise that the clusters represent the modes of the density $p(\mathbf{x})$, and the aim of clustering is to detect the high-density regions in the data. The modal structure of $p(\mathbf{x})$ can be summarized in a *cluster tree*. Each level in the cluster tree represents the feature space $L(\gamma, p) = \{\mathbf{x} \mid p(\mathbf{x}) > \gamma\}$. Cluster trees can be constructed using the single-link clustering algorithm to build neighborhood graphs, and finding the connected components in the neighborhood graphs. Density-based partitional clustering algorithms such as DB-

SCAN and OPTICS, are specialized non-parametric clustering techniques, which find the modes at a fixed user-defined density threshold. Mean-shift clustering algorithms estimate the density locally at each $\mathbf{x}$, and find the modes using a gradient ascent procedure on the local density.

1.2.2 Challenges in Data Clustering

Data clustering is a difficult problem, as reflected by the hundreds of clustering algorithms that have been published, and the new ones that continue to appear. Due to the inherent unsupervised nature of clustering, there are several factors that affect the clustering process.

- *Data representation.* The data can be input to clustering algorithms in two forms: (i) the $n \times d$ *pattern matrix* containing the d feature values for each of the n objects, and (ii) the $n \times n$ *proximity matrix*, whose entries represent the similarity/dissimilarity between the corresponding objects. Given a suitable similarity measure, it is easy to convert a pattern matrix to the proximity matrix. Similarly, methods like singular value decomposition and multi-dimensional scaling can been used to approximate the pattern matrix corresponding to the given proximity matrix [47]. Conventionally, hierarchical clustering algorithms assume input in the form of the proximity matrix, whereas partitional clustering algorithms accept the pattern matrix as input.

  The features used to represent the data in the pattern matrix play an important role in clustering. If the representation is good, the clustering algorithm will be able to find compact clusters in the data. Dimensionality of the data set is also crucial to the quality of clusters obtained. High-dimensional representations with redundant and noisy features not only lead to long clustering times, but may also deteriorate the cluster structure in the data. Feature selection and extraction techniques such as forward/backward selection and principal component analysis are used to determine the most discriminative features, and reduce the dimensionality of the data set [89]. Deep learning techniques [20] and kernel learning techniques [112]

can be employed to learn the data representation from the given data set.

- *Number of clusters.* Most clustering algorithms require the specification of the number of clusters C. While centroid-based, model-based and graph-theoretic algorithms directly accept the number of clusters as input, density and grid-based algorithms accept other parameters such as the maximum inter-cluster distance, which are indirectly related to the number of clusters. Automatically determining the number of clusters is a difficult problem and, in practice, domain knowledge is used to determine this parameter. Several heuristics have been proposed to estimate the number of clusters. In [172], the number of clusters is determined by minimizing the "gap" between the clustering error[10] for each value of C, and the expected clustering error of a reference distribution. Cross-validation techniques can be used to find the value of C at which the error curve corresponding to the validation data exhibits a sharp change [68].

- *Clustering Algorithm.* The objective of clustering dictates the algorithm chosen for clustering, and in turn, the quality and the structure of the clusters obtained. Centroid-based clustering algorithms such as *k*-means aim at minimizing the sum of the distances between the points and their representative centroids. This objective is suitable for applications where the clusters are compact and hyper-spherical or hyper-ellipsoidal. Density based algorithms aim at finding the dense regions in the data. The single-link hierarchical clustering algorithm finds long elongated clusters called "chains", as the criterion for merging clusters is local, whereas the complete-link hierarchical clustering algorithm finds large compact clusters. Each clustering algorithm is associated with a different similarity measure.

- *Similarity measures.* The similarity measure employed by the clustering algorithm is crucial to the structure of the clusters obtained. The choice of the similarity function depends on the data representation scheme, and the objective of clustering. A popular distance function is

[10] Refer to Section 1.3.1 for the definition of clustering error.

the squared Euclidean distance defined by

$$\mathbf{d}^2(\mathbf{x}_a, \mathbf{x}_b) = ||\mathbf{x}_a - \mathbf{x}_b||_2^2, \tag{1.1}$$

where $\mathbf{x}_a, \mathbf{x}_b \in \mathcal{D}$. However, the Euclidean distance is not suitable for all applications. Other distance measures such as Mahalanobis, Minkowski and non-linear distance measures have been applied in the literature to improve the clustering performance in many applications [171] (See Section 1.4).

- *Clustering Tendency, Quality and Stability.* Most clustering algorithms will find clusters in the given data set, even if the data does not contain any natural clusters. The study of clustering tendency deals with examining the data before executing the clustering algorithm, to determine if the data contains any clusters. Clustering tendency is usually assessed through visual assessment techniques which reorder the similarity matrix to examine whether or not the data contains clusters [85]. These techniques can also be used to determine the number of clusters in the data set.

  After obtaining the clusters, we need to evaluate the validity and quality of the clusters. Several measures have been identified to evaluate the clusters obtained, and the choice of the quality criterion depends on the application. Cluster validity measures are broadly classified as either internal or external measures [88]. Internal measures such as the value of the clustering algorithm's objective function and the inter-cluster distances assess the similarity between the cluster structure and the data. As clustering is an unsupervised task, it is logical to employ internal measures to evaluate the partitions. However, these measures are difficult to interpret and often vary from one clustering algorithm to another. On the other hand, external measures such as prediction accuracy and cluster purity use prior information like the true class labels to assess the cluster quality. External measures are more popularly used to evaluate and compare the clustering results of different clustering algorithms, as they are

easier to interpret than internal validity measures.

Cluster stability measures the sensitivity of the clusters to small perturbations in the data set [119]. It is dependent on both the data set and the algorithm used to perform clustering. Clustering algorithms which generate stable clusters are preferred as they will be robust to noise and outliers in the data. Stability is typically measured using data resampling techniques such as bootstrapping. Multiple data sets of the same size, generated from the same probability distribution, are clustered using the same algorithm and the similarity between the partitions of these data sets is used as a measure of the algorithm's stability.

- *Scalability.* In addition to the cluster quality, the choice of the clustering algorithm is also determined by the scalability of the algorithm. This factor becomes all the more crucial when designing systems for big data analysis. Two important factors that determine the scalability of a clustering algorithm are its running time complexity and its memory footprint. Clustering algorithms which have linear or sub-linear running time complexity, and require minimum amount of memory are desirable.

1.3 Clustering Big Data

When the size of the data set n is in the order of billions and the dimensionality of the data d is in the order of thousands, as is the case in many big data analytics problems, the scalability of the algorithm becomes an important factor while choosing a clustering algorithm. Hierarchical clustering algorithms are associated with at least $O(n^2d + n^2 \log(n))$ running time and $O(n^2)$ memory complexity, which renders them infeasible for large data sets. The same holds for many of the partitional clustering algorithms such as the model based algorithms like Latent Dirichlet Allocation, graph-based algorithms such as spectral clustering and density-based algorithms like DBSCAN. They have running time complexities ranging from $O(n \log(n))$ to $O(n^3)$ in terms of the number of points in the data, and at least linear time complexity with respect to the dimensionality

Table 1.2 Clustering techniques for Big Data.

Clustering approaches		Running time complexity	Memory complexity
Linear clustering	*k*-means	$O(nCd)$	$O(nd)$
Sampling-based clustering with sample size $m \ll n$	CLARA [94]	$O(Cm^2 + C(n-C))$	$O(n^2)$
	CURE [80]	$O(m^2 \log(m))$	$O(md)$
	Coreset [82]	$O(n + Cpolylog(n))$	$O(nd)$
Compression	BIRCH [197]	$O(nd)$	$M^\dagger$
	CLARANS [136]	$O(n^2)$	$O(n^2)$
Stream clustering	Stream [79], ClusTree [98]	$O(nCd)$	$M^\dagger$
	Scalable *k*-means [30], Single-pass *k*-means [62]	$O(nd)$	
	StreamKM++ [6]	$O(dns)$*	$O(ds \log(n/s))$*
Distributed clustering with P tasks	Parallel *k*-means [60, 199]	$O(nCd)$	$O(PC^2n^\delta),\ \delta > 0$
	MapReduce based spectral clustering [35]	$O(n^2d/P + r^3 + nr + nC^2)$**	$O(n^2/P)$
	Nearest-neighbor clustering [115]	$O(n \log(n)/P)$	$O(n/P)$

*$s = O(dC \log(n) \log^{d/2}(C \log(n)))$

** r represents the the rank of the affinity matrix

†M is a user-defined parameter representing the amount of memory available

d and the number of clusters C. Several clustering algorithms have been modified and special algorithms have been developed in the literature, to scale up to large data sets. Most of these algorithms involve a preprocessing phase to compress or distribute the data, before clustering is performed. Some of the popular methods to efficiently cluster large data sets (listed in Table 1.2) can be classified based on their preprocessing approach, as follows:

- Sampling-based methods reduce the computation time by first choosing a subset of the given data set and then using this subset to find the clusters. The key idea behind all sampling-based

clustering techniques is to obtain the cluster representatives, using only the sampled subset, and then assign the remaining data points to the closest representative. The success of these techniques depends on the premise that the selected subset is an unbiased sample and is representative of the entire data set. This subset is chosen either randomly (CLARA [94], CURE [80]) or through an intelligent sampling scheme such as coreset sampling [82, 183]. Coreset-based clustering first finds a small set of weighted data points called the coreset, which approximates the given data set, within a user-defined error margin, and then obtains the cluster centers using this coreset. In [63], it is proved that a coreset of size $O(C^2/\epsilon^4)$ is sufficient to obtain an $O(1+\epsilon)$ approximation, where ϵ is the error parameter.

- Clustering algorithms such as BIRCH [197] and CLARANS [136] improve the clustering efficiency by encapsulating the data set into special data structures like trees and graphs for efficient data access. For instance, BIRCH defines a data structure, called the Clustering-Feature Tree (CF-Tree). Each leaf node in this tree summarizes a set of points whose inter-point distances are less than a user-defined threshold, by the sum of the points, sum of the squares of the data points, and the number of points. Each non-leaf node summarizes the same statistics for all its child nodes. The points in the data set are added incrementally to the CF-Tree. The leaf entries of the tree are then clustered using an agglomerative hierarchical clustering algorithm to obtain the final data partition. Other approaches summarize the data into *kd*-trees and R-trees for fast *k*-nearest neighbor search [115].

- Stream clustering [8] algorithms are designed to operate in a single pass over an arbitrary-sized data set. Only the sufficient statistics (such as the mean and variance of the clusters, when the clusters are assumed to be drawn from a Gaussian mixture) of the data seen so far are retained, thereby reducing the memory requirements. One of the first stream clustering algorithms was proposed by Guha *et al.* [79]. They first summarize the data stream into a larger number of clusters than desired, and then cluster the centroids obtained in the first step.

Stream clustering algorithms such as CluStream [8], ClusTree [98], scalable *k*-means [30], and single-pass *k*-means [62] were built using a similar idea, containing an online phase to summarize the incoming data, and an offline phase to cluster the summarized data. The summarization is usually in the form of trees [8, 30], grids [32, 36] and coresets [6, 63]. For instance, the CluStream algorithm summarizes the data set into a CF-Tree, in which each node stores the linear sum and the squared sum of a set of points which are within a user-defined distance from each other. Each node represents a micro-cluster whose center and radius can be found using the linear and squared sum values. The *k*-means algorithm is the algorithm of choice for the offline phase to obtain the final clusters.

- With the evolution of cloud computing, parallel processing techniques for clustering have gained popularity [48, 60]. These techniques speed up the clustering process by first dividing the task into a number of independent sub-tasks that can be performed simultaneously, and then efficiently merging these solutions into the final solution. For instance, in [60], the MapReduce framework [148] is employed to speed up the *k*-means and the *k*-medians clustering algorithms. The data set is split among many processors and a small representative data sample is obtained from each of the processors. These representative data points are then clustered to obtain the cluster centers or medians. In parallel latent Dirichlet allocation, each task finds the latent variables corresponding to a different component of the mixture [133]. The Mahout platform [143] implements a number of parallel clustering algorithms, including parallel *k*-means, latent Dirichlet allocation, and mean-shift clustering [37, 133, 135, 199]. Billions of images were clustered using an efficient parallel nearest-neighbor clustering in [115].

Data sets of sizes close to a billion have been clustered using the parallelized versions of the *k*-means, nearest neighbor and spectral clustering algorithms. To the best of our knowledge, based on the published articles, the largest data set that has been clustered consisted of a 1.5 billion images,

each represented by a 100-dimensional vector containing the Haar wavelet coefficients [115]. They were clustered into 50 million clusters using the distributed nearest neighbor algorithm in 10 hours using $2,000$ CPUs. Data sets that are big in both size (n) and dimensionality (d), like social-network graphs and web graphs, were clustered using subspace clustering algorithms and parallel spectral clustering algorithms [35, 181].

1.3.1 Clustering with *k*-means

Among the various $O(n)$ running time clustering algorithms in Table 1.2, the most popular algorithm for clustering large scale data sets is the *k-means* algorithm [87]. It is easy to implement, simple and efficient. It is easy to parallelize, has relatively few parameters when compared to the other algorithms, and yields clustering results similar to many other clustering algorithms [192]. Millions of points can be clustered using *k*-means within minutes. Extensive research has been performed to solve the *k*-means problem and obtain strong theoretical guarantees with respect to its convergence and accuracy. For these reasons, we focus on the *k*-means algorithm in this thesis.

The key idea behind *k*-means is to minimize the *clustering error*, defined as the sum of the squared distances between the data points and the center of the cluster to which each point is assigned. This can be posed as the following min-max optimization problem:

$$\min_{U \in \mathcal{P}} \max_{\mathbf{c}_k \in \chi} \sum_{k=1}^{C} \sum_{i=1}^{n} U_{k,i} \mathbf{d}^2 \left(\mathbf{c}_k, \mathbf{x}_i \right), \tag{1.2}$$

where $U = (\mathbf{u}_1, \ldots, \mathbf{u}_C)^\top$ is the cluster membership matrix, $\mathbf{c}_k \in \chi, k \in [C]$ are the cluster centers, and domain $\mathcal{P} = \{U \in \{0,1\}^{C \times n} : U^\top \mathbf{1} = \mathbf{1}\}$, where $\mathbf{1}$ is a vector of all ones. The most commonly used distance measure $\mathbf{d}(\cdot, \cdot)$ is the squared Euclidean distance measure, defined in (1.1). The *k*-means problem with the squared Euclidean distance measure is defined as

$$\min_{U \in \mathcal{P}} \max_{\mathbf{c}_k \in \chi} \sum_{k=1}^{C} \sum_{i=1}^{n} U_{k,i} \left|\left| \mathbf{c}_k - \mathbf{x}_i \right|\right|_2^2. \tag{1.3}$$

Algorithm 1 *k*-means

Input:
- $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}, \mathbf{x}_i \in \Re^d$: the set of n d-dimensional data points to be clustered
- C: the number of clusters

Output: Cluster membership matrix $U \in \{0,1\}^{C \times n}$

Randomly initialize the membership matrix U with zeros and ones, ensuring that $U^\top \mathbf{1} = \mathbf{1}$.

repeat

  Compute the cluster centers $\mathbf{c}_k = \frac{1}{\mathbf{u}_k^\top \mathbf{1}} \sum_{i=1}^{n} U_{k,i}\mathbf{x}_i, k \in [C]$.

  for $i = 1, \ldots, n$ **do**

    Find the closest cluster center k_* for $\mathbf{x}_i$, by solving

$$k_* = \underset{k \in [C]}{\operatorname{argmin}} ||\mathbf{c}_k - \mathbf{x}_i||_2^2 .$$

    Update the i^{th} column of U by $U_{k,i} = 1$ for $k = k_*$ and zero, otherwise.

  end for

until convergence is reached

The above problem (1.3) is an NP-complete integer programming problem, due to which it is difficult to solve [121]. A greedy approximate algorithm, proposed by Lloyd solves (1.3) iteratively [116]. The centers are initialized randomly. In each iteration, every data point is assigned to the cluster whose center is closest to it, and then the cluster centers are recalculated as the means of the points assigned to the cluster, i.e. the k^{th} center $\mathbf{c}_k$ is obtained as

$$\mathbf{c}_k = \frac{1}{n_k} \sum_{i=1}^{n} U_{k,i}\mathbf{x}_i, k \in [C], \tag{1.4}$$

where $n_k = \mathbf{u}_k^\top \mathbf{1}$ is the number of points assigned to the k^{th} cluster. These two steps are repeated until the cluster labels of the data points do not change in consecutive iterations. This procedure is described in Algorithm 1. It has $O(ndCl)$ running time complexity and $O(nd)$ memory complexity, where l is the number of iterations required for convergence. Several methods have been developed in the literature to initialize the algorithm intelligently and ensure that the solution obtained is a $(1+\epsilon)$-approximation of the optimal solution of (1.3) [12, 101].

1.4 Kernel Based Clustering

The issue of scalability can be addressed by using the large scale clustering algorithms described in Section 1.3. However, most of these algorithms, including *k*-means, are linear clustering algorithms, i.e. they assume that the clusters are linearly separable in the input space (e.g. the data set shown in Figure 1.3(a)) and define the inter-point similarities using measures such as the Euclidean distance. They suffer from the following two main drawbacks:

(i) Data sets that contain clusters that cannot be separated by a hyperplane in the input space cannot be clustered by linear clustering algorithms. For this reason, all the clustering algorithms in Table 1.2, with the exception of spectral clustering, are only able to find compact well-separated clusters in the data. They are also not robust to noise and outliers in the data. Consider the example shown in Figure 1.4. The data set in Figure 1.4(a) contains 500 points in the form of two semi-circles. We expect a clustering algorithm to group the points in each semi-circle, and detect the two semi-circular clusters. The clusters resulting from *k*-means with Euclidean distance are shown in Figure 1.4(b). Due to the use of Euclidean distance, the two-dimensional space is divided into two half-spaces and the resulting clusters are separated by the black dotted line. Other Euclidean-distance based partitional algorithms also find similar incorrect partitions.

(ii) Non-linear similarity measures can be used to find arbitrarily shaped clusters, and are more suitable for real-world applications. For example, suppose two images are represented by their pixel intensity values. The images may be considered more similar to each other if they comprise of similar pixel values, as shown in Figure 1.5. Thus the difference between the images is reflected better by the dissimilarity of image histograms than by the Euclidean distance between the pixel values [14, 106].

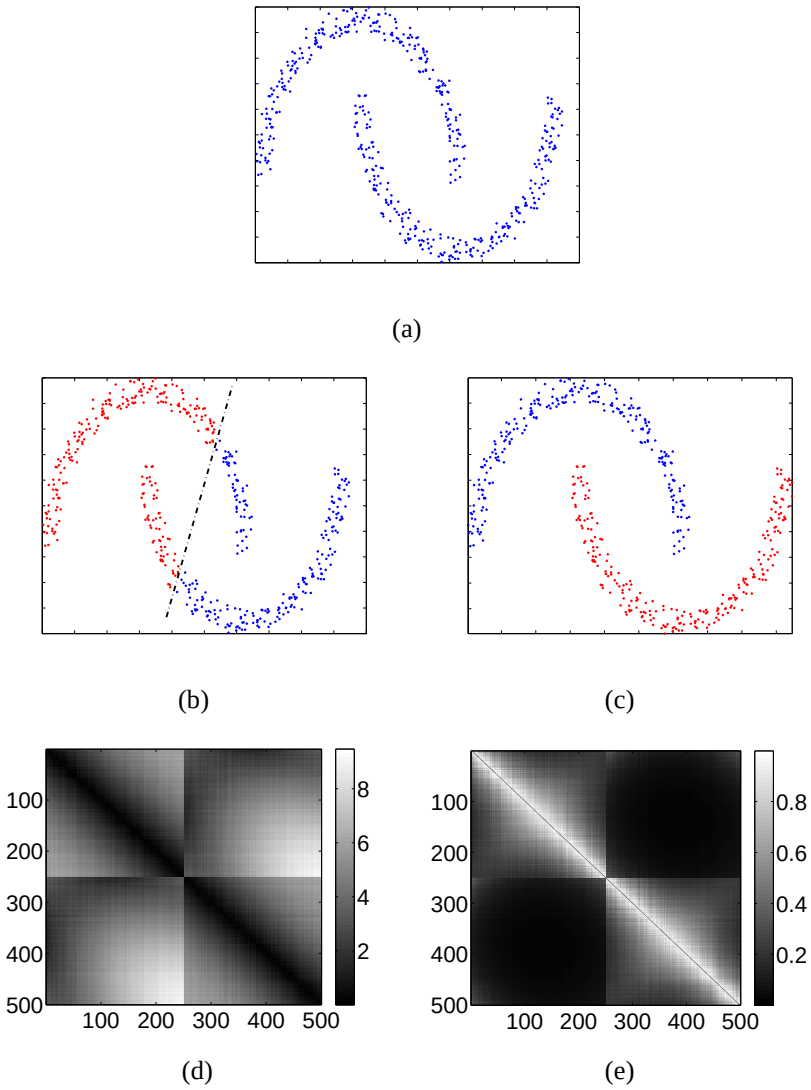


Figure 1.4 A two-dimensional example that demonstrates the limitations of *k*-means clustering. 500 two-dimensional points containing two semi-circular clusters are shown in Figure (a). Points numbered $1-250$ belong to the first cluster and points numbered $251-500$ belong to the second cluster. The clusters obtained using *k*-means (using Euclidean distance measure) do not reflect the true underlying clusters (shown in Figure (b)), because the clusters are not linearly separable as expected by the *k*-means algorithm. On the other hand, the kernel *k*-means algorithm using the RBF kernel (with kernel width $\sigma^2 = 0.4$) reveals the true clusters (shown in Figure (c)). Figures (d) and (e) show the 500×500 similarity matrices corresponding to the Euclidean distance and the RBF kernel similarity, respectively. The RBF kernel similarity matrix contains distinct blocks which distinguish between the points from different clusters. The similarity between the points in the same true cluster is higher than the similarity between points in different clusters. The Euclidean distance matrix, on the other hand, does not contain such distinct blocks, which explains the failure of the *k*-means algorithm on this data.

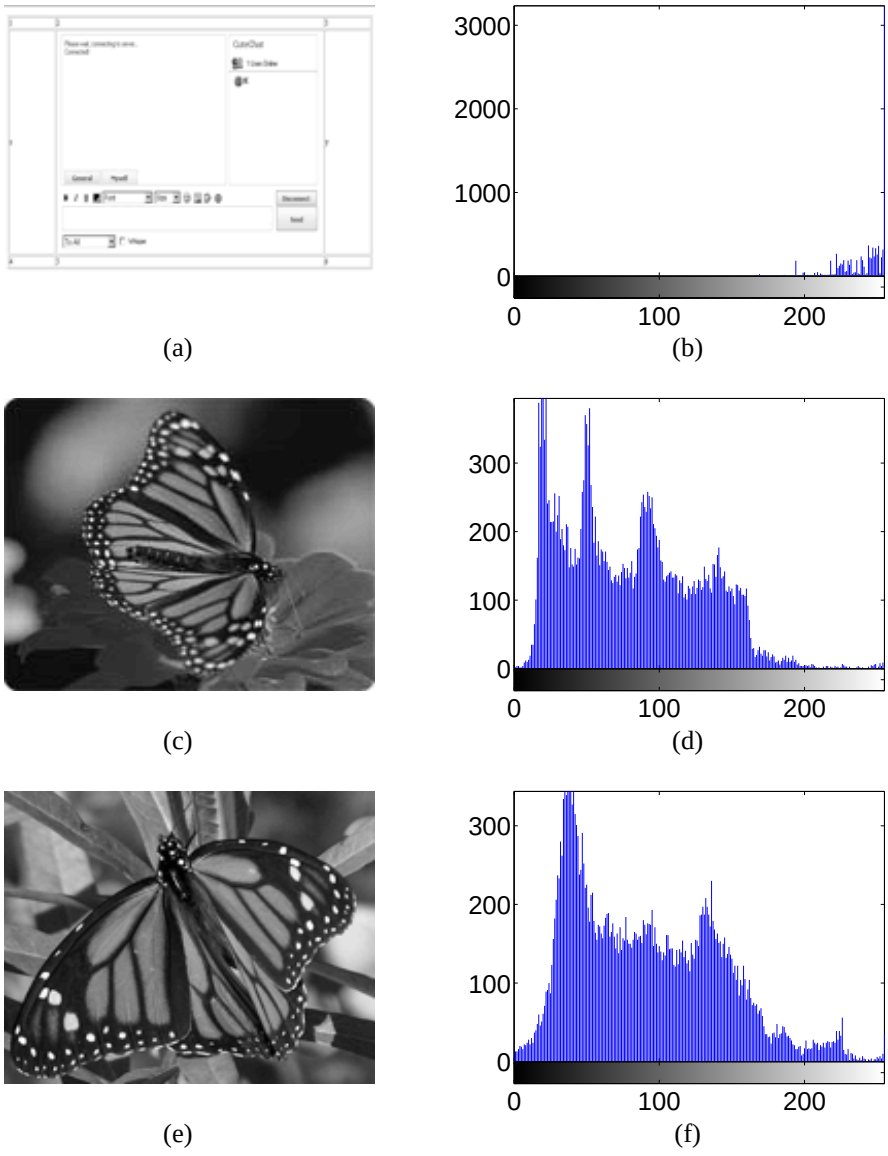


Figure 1.5 Similarity of images expressed through gray level histograms. The histogram of the intensity values of the image of a website (Figure (b)) is very different from the histograms of the images of butterflies (Figures (d) and (f)). The histograms of the two butterfly images are similar to each other.

The issue of non-linear separability is tackled using *kernel functions*[11]. The key behind the success of kernel-based learning algorithms is the fact that any data set becomes linearly separable when projected to an appropriate high dimensional space. Consider a non-linear function $\varphi : \chi \rightarrow \mathcal{H}_\kappa$, which maps the points in the *input space* χ to a high dimensional *feature space* $\mathcal{H}_\kappa$. The distance between the data points in this feature space can be defined in terms of the dot products of the projected points. For instance, the Euclidean distance between two points $\mathbf{x}_a$ and $\mathbf{x}_b$ in $\mathcal{H}_\kappa$ is defined as

$$||\varphi(\mathbf{x}_a) - \varphi(\mathbf{x}_b)||_2^2 = \langle\varphi(\mathbf{x}_a), \varphi(\mathbf{x}_a)\rangle + \langle\varphi(\mathbf{x}_b), \varphi(\mathbf{x}_b)\rangle - 2\langle\varphi(\mathbf{x}_a), \varphi(\mathbf{x}_b)\rangle.$$

In practical applications, the dimensionality of $\mathcal{H}_\kappa$ is extremely high, possibly infinite. Hence, the explicit computation of the mapping φ is highly computationally intensive and, in most cases, infeasible. This computation is avoided by replacing the dot product with a non-linear kernel distance function $\kappa(\cdot, \cdot) : \chi \times \chi \rightarrow \Re$. The distance between any two points is now defined in terms of the kernel function κ as

$$\mathbf{d}_\kappa^2(\mathbf{x}_a, \mathbf{x}_b) = \kappa(\mathbf{x}_a, \mathbf{x}_a) + \kappa(\mathbf{x}_b, \mathbf{x}_b) - 2\kappa(\mathbf{x}_a, \mathbf{x}_b). \tag{1.5}$$

A kernel function κ is admissible if and only if it satisfies the Mercer's condition [159, Theorem 2.10]. Informally stated, Mercer's theorem asserts that there exists a mapping φ and an expansion $\kappa(\mathbf{x}_a, \mathbf{x}_b) = \varphi(\mathbf{x}_a)^\top\varphi(\mathbf{x}_b)$ if and only if, for any function $g(\mathbf{x})$ such that $\int g(\mathbf{x})^2 d\mathbf{x}$ is finite, we have

$$\int \kappa(\mathbf{x}_a, \mathbf{x}_b)g(\mathbf{x}_a)g(\mathbf{x}_b)d\mathbf{x}_a d\mathbf{x}_b \geq 0.$$

Such a kernel is known as the Mercer kernel or Reproducing Kernel, and the feature space $\mathcal{H}_\kappa$ is called the *Reproducing Kernel Hilbert Space (RKHS)*. The matrix $K = [\kappa(\mathbf{x}_i, \mathbf{x}_j)], \mathbf{x}_i, \mathbf{x}_j \in \mathcal{D}$ is

[11] http://crsouza. blogspot.com/2010/03/kernel-functions-for-machine-learning. html

Table 1.3 Popular kernel functions.

Linear	$\kappa(\mathbf{x}_a, \mathbf{x}_b) = \mathbf{x}_a^\top \mathbf{x}_b + c$ for constant c
Polynomial	$\kappa(\mathbf{x}_a, \mathbf{x}_b) = \left(\mathbf{x}_a^\top \mathbf{x}_b + c\right)^d$, d is the degree of the polynomial kernel
RBF	$\kappa(\mathbf{x}_a, \mathbf{x}_b) = \exp\left(-\frac{\|\|\mathbf{x}_a - \mathbf{x}_b\|\|_2^2}{2\sigma^2}\right)$, $\sigma > 0$ is the kernel width parameter
Laplacian	$\kappa(\mathbf{x}_a, \mathbf{x}_b) = \exp\left(-\frac{\|\|\mathbf{x}_a - \mathbf{x}_b\|\|}{\sigma}\right)$
Chi-square	$\kappa(\mathbf{x}_a, \mathbf{x}_b) = 1 - \sum_{i=1}^{d} \frac{\left(\mathbf{x}_a^i - \mathbf{x}_b^i\right)^2}{0.5\left(\mathbf{x}_a^i + \mathbf{x}_b^i\right)}$
Histogram Intersection	$\kappa(\mathbf{x}_a, \mathbf{x}_b) = \sum \min\left(\text{hist}\left(\mathbf{x}_a\right), \text{hist}\left(\mathbf{x}_b\right)\right)$
String kernel	Number of common subsequences between string sequences $\mathbf{x}_a$ and $\mathbf{x}_b$

known as the kernel matrix or Gram matrix. The simplest kernel functions are positive definite kernels whose corresponding kernel matrix is Hermitian and positive-definite. The Radial Basis Function (RBF) kernel defined by

$$\kappa(\mathbf{x}_a, \mathbf{x}_b) = \exp\left(-\frac{||\mathbf{x}_a - \mathbf{x}_b||_2^2}{2\sigma^2}\right), \sigma > 0 \tag{1.6}$$

is a popular positive-definite kernel function. It performs well on a large number of benchmark data sets. The parameter σ^2, known as the kernel width, scales the distance between the points. Table 1.3 lists some of the popular kernel functions. Chi-square kernel, histogram intersection kernel and their variants are commonly used in image and video-related applications. String kernels are popular in text-mining applications. The remaining kernels in Table 1.3 are generic kernels. Using the linear kernel is the same as using the Euclidean distance measure.

Kernel based clustering techniques use (1.5) to define the similarity between objects. Consequently, when provided with the appropriate kernel function, they have the ability to capture the non-linear structure in real world data sets and, thus, usually perform better than the linear clustering algorithms, in terms of cluster quality [95]. Various kernel-based clustering algorithms have been developed, including kernel *k*-means, spectral clustering, support vector clustering, maximum margin clustering, kernel self-organizing maps and kernel neural gas [65].

Spectral clustering [118] is based on the idea of spectral graph partitioning. The data points are represented as nodes in a graph and the affinity between the nodes is defined by the kernel similarity between the points. The graph is partitioned into C components by first computing the graph Laplacian matrix and the eigenvectors corresponding to its smallest C eigenvalues, and then clustering the eigenvectors into C clusters using *k*-means. The data partition is obtained via the graph partition. Spectral clustering is widely employed for image segmentation and graph partitioning problems.

Support vector clustering [19] involves projecting the data to a high dimensional feature space and searching for a minimum enclosing sphere in this space. This enclosing sphere is projected back into the input space and the support vectors are used to define the cluster boundaries. All points that lie within a cluster boundary are assigned to the same cluster. The maximum margin clustering technique [190] finds the cluster labeling which when used to find a maximum margin classifier (e.g. Support Vector Machines) for the given data, results in a margin that is maximal over all possible cluster labelings. A convex optimization problem with the cluster labels and the margin of the Support Vector Machine as variables, and constraints on the number of points per cluster and the difference in the cluster sizes, is formulated. The labels and the classifier margin are optimized simultaneously to find the optimal cluster labels.

The kernel self-organizing map [120] algorithm extends the self-organizing map [96] algorithm to use kernel distance measures. The key idea behind this algorithm is to construct a low-dimensional (typically two-dimensional) topology-preserving map of the input data set through competitive learning. A self-organizing map, also known as the Kohenen map, consists of a two-layer network, the input layer containing d nodes and an output layer containing at least C nodes. Each output node is randomly initialized with a weight. When a new data point is input to the network, the node whose weight is closest to the input data point in terms of the kernel distance is determined. This node is called the Best Matching Unit. The weights of the best matching unit and its neighboring nodes are updated, based on a pre-defined neighborhood function. After a number

of passes over the data set, the weights of the nodes converge to form distinctive regions in the output layer from which the clusters in the data can be read off.

The kernel neural gas algorithm [145], inspired by the self-organizing map algorithm, also creates a map of the input data. The difference between the two methods is that while, only the weights of a few neighboring nodes of the best matching unit are updated in a self-organizing map, the weights of all the nodes are updated in the neural gas algorithm. The nodes are ranked based on their proximity to the best matching unit, and their weights updated on the basis of their rank. The nearest node is updated by a higher factor than the farthest node. This update mechanism leads to neural gas converging faster than self-organizing maps.

Similar to the *k*-means algorithm, the kernel *k*-means algorithm [160] is the most popular kernel-based clustering algorithm due to it simplicity. Several studies have also established the theoretical equivalence of kernel *k*-means and other kernel-based clustering methods, suggesting that they yield similar results [51, 52].

1.4.1 Kernel *k*-means

The kernel *k*-means algorithm can be viewed as a non-linear extension of the *k*-means algorithm. It replaces the Euclidean distance function (1.1) employed in the *k*-means algorithm with a non-linear kernel distance function defined in (1.5).

Let $K \in \Re^{n \times n}$ be the kernel matrix with $K_{i,j} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$, where $\kappa(\cdot, \cdot)$ is the kernel function. Let $\mathcal{H}_\kappa$ be the Reproducing Kernel Hilbert Space (RKHS) endowed by the kernel function $\kappa(\cdot, \cdot)$, and $||\cdot||_{\mathcal{H}_\kappa}$ be the functional norm for $\mathcal{H}_\kappa$. Similar to the *k*-means problem, the objective of kernel *k*-means is to minimize the clustering error. Hence, the kernel *k*-means problem can be cast as the following optimization problem:

$$\min_{U \in \mathcal{P}} \max_{\{\mathbf{c}_k(\cdot) \in \mathcal{H}_\kappa\}_{k=1}^{C}} \sum_{k=1}^{C} \sum_{i=1}^{n} U_{k,i} \|\mathbf{c}_k(\cdot) - \kappa(\mathbf{x}_i, \cdot)\|_{\mathcal{H}_\kappa}^2, \tag{1.7}$$

Algorithm 2 Kernel *k*-means

Input:
- $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}, \mathbf{x}_i \in \Re^d$: the set of n d-dimensional data points to be clustered
- $\kappa(\cdot, \cdot) : \chi \times \chi \mapsto \Re$: the kernel function
- C: the number of clusters

Output: Cluster membership matrix $U \in \{0, 1\}^{C \times n}$

Compute the kernel matrix $K = [\kappa(\mathbf{x}_i, \mathbf{x}_j)]_{n \times n}$.

Randomly initialize the membership matrix U with zeros and ones, ensuring that $U^\top \mathbf{1} = \mathbf{1}$.

repeat

  for $i = 1, \ldots, n$ **do**

    Find the closest cluster center k_* for $\mathbf{x}_i$, by solving

$$\begin{aligned} k_* &= \underset{k \in [C]}{\operatorname{argmin}} \, ||\mathbf{c}_k(\cdot) - \kappa(\mathbf{x}_i, \cdot)||^2_{\mathcal{H}_\kappa} \\ &= \underset{k \in [C]}{\operatorname{argmin}} \, \frac{\mathbf{u}_k^\top K \mathbf{u}_k}{\left(\mathbf{u}_k^\top \mathbf{1}\right)^2} - 2\frac{\mathbf{u}_k^\top K_i}{\mathbf{u}_k^\top \mathbf{1}}, \end{aligned}$$

    where $\mathbf{u}_k$ is the k^{th} column of $U^\top$, and K_i is the i^{th} column of K.

    Update the i^{th} column of U by $U_{k,i} = 1$ for $k = k_*$ and zero otherwise.

  end for

until convergence is reached

where $U = (\mathbf{u}_1, \ldots, \mathbf{u}_C)^\top$ is the cluster membership matrix, $\mathbf{c}_k(\cdot) \in \mathcal{H}_\kappa, k \in [C]$ are the cluster centers, and domain $\mathcal{P} = \{U \in \{0, 1\}^{C \times n} : U^\top \mathbf{1} = \mathbf{1}\}$, where $\mathbf{1}$ is a vector of all ones. The above problem is also NP-complete. A simplified version of the problem, which relaxes the constraints on U, is solved to obtain the solution [72, 192]. Let $n_k = \mathbf{u}_k^\top \mathbf{1}$ be the number of data points assigned to the k^{th} cluster, and

$$\begin{aligned} \widehat{U} &= (\widehat{\mathbf{u}}_1, \ldots, \widehat{\mathbf{u}}_C)^\top = [\text{diag}(n_1, \ldots, n_C)]^{-1} U, \\ \widetilde{U} &= (\widetilde{\mathbf{u}}_1, \ldots, \widetilde{\mathbf{u}}_C)^\top = [\text{diag}(\sqrt{n_1}, \ldots, \sqrt{n_C})]^{-1} U, \end{aligned} \tag{1.8}$$

denote the ℓ_1 and ℓ_2 normalized membership matrices, respectively.

It is easy to verify that, given the $C \times n$ cluster membership matrix U, the optimal solution for

the cluster centers is

$$\mathbf{c}_k(\cdot) = \sum_{i=1}^{n} \widehat{U}_{k,i}\kappa(\mathbf{x}_i, \cdot), k \in [C]. \tag{1.9}$$

As a result, we can formulate (1.7) as the following optimization problem over U:

$$\min_{U \in \mathcal{P}} \text{tr}(K) - \text{tr}(\widetilde{U}K\widetilde{U}^\top), \tag{1.10}$$

which can be further reformulated as the following trace maximization problem:

$$\max_{U \in \mathcal{P}} \text{tr}(\widetilde{U}K\widetilde{U}^\top). \tag{1.11}$$

Note that the *k*-means optimization problem in (1.3) can also be written as the following trace maximization problem:

$$\max_{U \in \mathcal{P}} \text{tr}(\widetilde{U}XX^\top\widetilde{U}^\top), \tag{1.12}$$

where $X = (\mathbf{x}_1, \ldots, \mathbf{x}_n)^\top$ is the $n \times d$ pattern matrix corresponding to the data set $\mathcal{D}$. Therefore, a greedy iterative algorithm similar to the *k*-means algorithm can be employed to solve (1.11), with the Euclidean distance function replaced by the kernel similarity function.

The kernel *k*-means algorithm is described in Algorithm 2. Figure 1.4(c) shows the result of applying the kernel *k*-means algorithm to the synthetic semi-circles data set in Figure 1.4(a) using the RBF kernel function in (1.6), with the kernel width σ^2 set to 0.4. It can be observed that kernel *k*-means is able to detect the two semi-circles correctly, unlike the *k*-means algorithm.

1.4.2 Challenges

Though kernel based clustering algorithms achieve better cluster quality, they suffer from two major limitations.

Table 1.4 Comparison of the running times of *k*-means and kernel *k*-means on a 100-dimensional synthetic data set containing 10 clusters and exponentially increasing number of data points, on a 2.8 GHz processor with 40 GB memory.

Data set size		10^4	10^5	10^6	10^7	10^8
Running time in seconds	***k*-means**	0.03	0.17	2.30	34.90	5508.50
	Kernel *k*-means	3.09	320.10	> 48 hours		

1.4.2.1 Scalability

A naive implementation of kernel *k*-means requires the computation of the $n \times n$ kernel matrix K (Step 3 in Algorithm 2) which takes $O(n^2)$ time and memory. Clustering millions of objects using kernel *k*-means requires more than $8,000$ GB of memory and large amount of computing resources. Table 1.4 compares the running times of the *k*-means and the kernel *k*-means algorithms on a 100-dimensional synthetic data set containing 10 clusters and exponentially increasing number of points. The algorithms were executed on a 2.8 GHz processor with 40 GB memory. It can be seen that running kernel *k*-means is far more expensive than running *k*-means, especially on large data sets.

It is also expensive to assign previously unseen data points to clusters using kernel *k*-means, often termed as the *out-of-sample-problem*. To find the cluster label for a new data point $\mathbf{x}$, we need to compute the distance between $\mathbf{x}$ and all the cluster centers as follows:

$$\begin{aligned} d_\kappa^2(\mathbf{x}, \mathbf{c}_k) &= ||\mathbf{c}_k(\cdot) - \kappa(\mathbf{x}, \cdot)||_{\mathcal{H}_\kappa}^2 \\ &= \frac{\mathbf{u}_k^\top K \mathbf{u}_k}{\left(\mathbf{u}_k^\top \mathbf{1}\right)^2} - 2\frac{\mathbf{u}_k^\top K_{\mathbf{x}}}{\mathbf{u}_k^\top \mathbf{1}}, k \in [C], \end{aligned} \tag{1.13}$$

where $K_{\mathbf{x}} = (\kappa(\mathbf{x}, \mathbf{x}_1), \ldots, \kappa(\mathbf{x}, \mathbf{x}_n))^\top$. It requires the computation of the $O(n)$-sized vector $K_{\mathbf{x}}$ in addition to the kernel matrix K. This is due to the fact that there is no explicit representation for the cluster centers. If there was a d-dimensional representation for the cluster centers $\mathbf{c}_k$ (as in the case of *k*-means), the distance $d_\kappa^2(\mathbf{x}, \mathbf{c}_k)$ can be computed in $O(d)$ time.

Clearly, scalability is a major challenge faced by kernel *k*-means. Other kernel-based algo-

rithms also have high running time complexity. For instance, spectral clustering involves the computation of the top C eigenvectors of the kernel matrix, which is of $O(n^3)$ complexity.

In the literature, the issue of scalability has largely been addressed through the use of cloud computing and parallel algorithms. The Mahout platform [143] implements the parallel spectral clustering algorithm which uses the distributed Lanczos eigensolver to obtain the eigenvectors of the Laplacian matrix [35]. Distributed implementations of Support Vector Machines have been developed to perform clustering [78, 169]. However, parallelization of kernel based algorithms is not simple due to their non-linear nature [15]. For instance, in order to parallelize kernel *k*-means, one must replicate the data to all the tasks, leading to large resource and communication overheads.

Approximate clustering techniques are useful in alleviating this issue. Sampling methods, such as the Nystrom method [187], have been employed to obtain low rank approximation of the kernel matrix to address this challenge [67, 113]. Low-dimensional projection combined with sampling have been used to further improve the clustering efficiency and tackle the out-of-sample problem [11, 153].

1.4.2.2 Choice of kernel

The role of the kernel function is to reflect the true structure of the data set. However, if the kernel function is chosen wrongly, the performance of the clustering algorithm degrades. The RBF kernel defined in (1.6) performs well on most benchmark data sets. However, even for the RBF kernel, the kernel width parameter has to be chosen carefully. Figure 1.6 demonstrates the sensitivity of kernel *k*-means to the kernel width parameter. Kernel *k*-means is executed on the semi-circles data set shown in Figure 1.4(a), using the RBF kernel with kernel width values: 0.4 and 0.1. The clusters obtained are shown in Figures 1.6(b) and 1.6(c) respectively. When $\sigma^2 = 0.4$, the true clusters are revealed. On the other hand, when $\sigma^2 = 0.1$, the clusters are distorted. Figure 1.6(d) plots the clustering error of kernel *k*-means, defined in (1.10), against the RBF kernel width. It is clear that the performance depends on the choice of the kernel width. Hence, another challenge associated

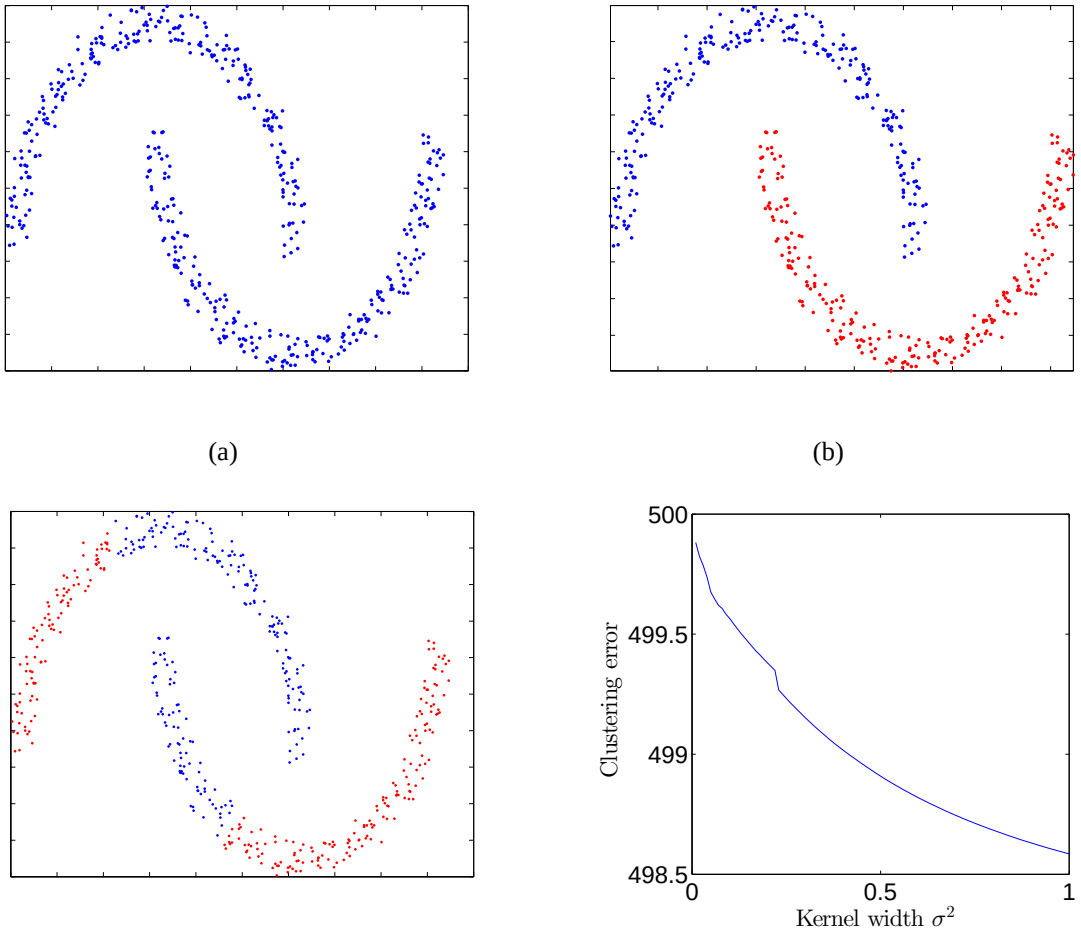


Figure 1.6 Sensitivity of the kernel *k*-means algorithm to the choice of kernel function. The semi-circles data set (shown in Figure (a)) is clustered using kernel *k*-means with the RBF kernel. When the kernel width is set to 0.4, the two clusters are correctly detected (shown in Figure (b)), whereas when the kernel width is set to 0.1, the points are clustered incorrectly (shown in Figure (c)). Figure (d) shows the variation in the clustering error of kernel *k*-means, defined in (1.10), with respect to the kernel width.

with kernel based algorithms is the choice of the kernel function and the kernel parameters.

Kernel learning techniques aim at learning a positive semi-definite kernel matrix that reflects the true similarity between the points in the data set [4]. In the supervised learning setting, the kernel is optimized to align with the true class structure of the data. This is achieved by either minimizing the error of the classifier for the chosen kernel, or maximizing the similarity between the kernel and the class matrix. As the class labels are not available in the setting of unsupervised learning, other criterion such as compactness of the clusters in the feature space, and degree of alignment with the structure of the data are utilized [112, 177, 200].

1.5 Thesis Contributions

The objective of this thesis is to design clustering algorithms that can accurately identify the clusters in data sets containing billions of points, thousands of features and thousands of clusters. As kernel-based clustering algorithms generally achieve high cluster quality, provided the correct kernel function is chosen, we address the scalability challenge associated with kernel based clustering algorithms. Our main contribution is the development of efficient approximations of the kernel *k*-means algorithm to enable kernel based clustering of large data sets. We demonstrate analytically and empirically that the proposed approximate algorithms are comparable to kernel *k*-means in terms of accuracy and, at the same time, comparable to *k*-means in terms of efficiency, achieving the desired trade-off between scalability and accuracy. We then extend the proposed approximate algorithms to handle distributed and streaming data, pushing the limits on the number of objects that can be clustered accurately with limited computing and memory resources. Figure 1.7 shows the scalability of some of the popular linear and kernel-based clustering algorithms in terms of n, d and C, and the contribution of the proposed clustering algorithms in improving the scalability of kernel-based clustering.

In the following, we describe the specific contributions of each chapter:

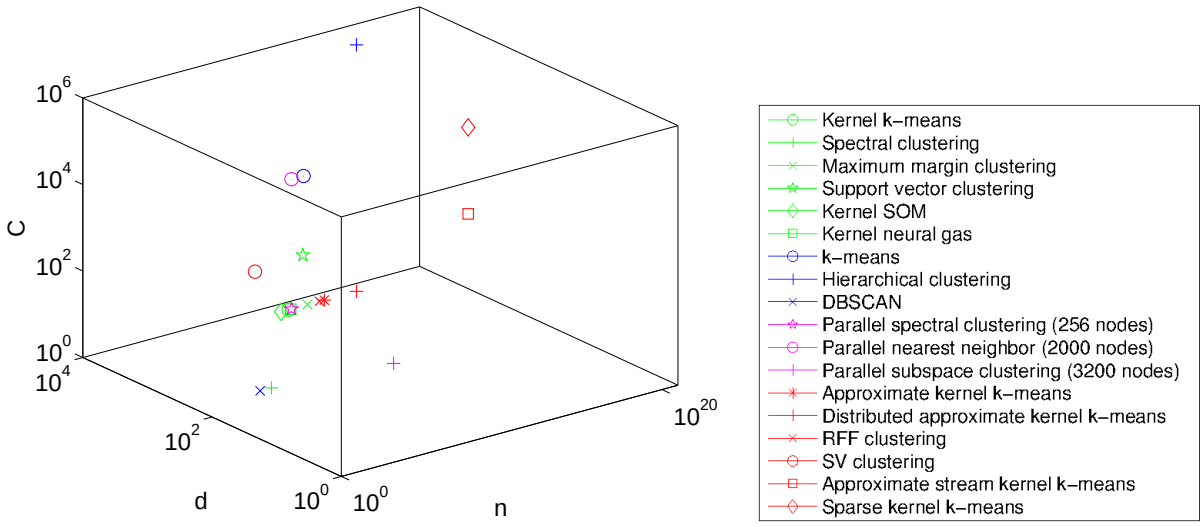


Figure 1.7 Scalability of clustering algorithms in terms of n, d and C, and the contribution of the proposed algorithms in improving the scalability of kernel-based clustering. The plot shows the maximum size of the data set that can be clustered with less than 100 GB memory on a 2.8 GHz processor with a reasonable amount of clustering time (less than 10 hours). The linear clustering algorithms are represented in blue, current kernel-based clustering algorithms are shown in green, parallel clustering algorithms are shown in magenta, and the proposed clustering algorithms are represented in red. Existing kernel-based clustering algorithms can cluster only up to the order of $10,000$ points with 100 features into 100 clusters. The proposed batch clustering algorithms (approximate kernel *k*-means, RFF clustering, and SV clustering algorithms) are capable of performing kernel-based clustering on data sets as large as 10 million, with the same resource constraints. The proposed online clustering algorithms (approximate stream kernel *k*-means and sparse kernel *k*-means algorithms) can cluster arbitrarily-sized data sets with dimensionality in the order of $1,000$ and the number of clusters in the order of $10,000$.

- Chapters 2 and 3 address the scalability of kernel *k*-means using kernel approximation techniques. The computational demand of kernel *k*-means stems from the fact that it computes an $n \times n$ kernel matrix K, leading to $O(n^2)$ running time and memory complexity. This can be alleviated by replacing the kernel matrix K with an approximate matrix which can be computed more efficiently. In Chapter 2, we first present a randomized algorithm, called *approximate kernel k-means* which replaces K, with a low rank approximate kernel matrix. Its complexity is linear in terms of n, while its clustering performance is equivalent to that of kernel *k*-means. We then extend the proposed approximate algorithm to handle large data sets in a distributed environment. In Chapter 3, we propose two clustering algorithms: *RFF clustering* and *SV clustering*, which employ *random feature maps* [92, 147] to obtain low-dimensional representations for the data points, such that the dot product of any two points in the low-dimensional space approximates the kernel similarity between them. This allows us to execute a linear clustering algorithm on the transformed data points. The SV clustering algorithm has a lower running time than the approximate kernel *k*-means algorithm. It also allows the explicit computation of the cluster centers, leading to an efficient solution to the out-of-sample clustering problem. We demonstrate that it is possible to cluster billions of data points efficiently and accurately using the algorithms proposed in these two chapters. For instance, we were able to cluster a synthetic data set containing 1 billion 10-dimensional points using the distributed approximate kernel *k*-means algorithm in 15 minutes (on a computing cluster with $1,024$, 2.8 GHz processors and shared 40 GB memory), with high cluster quality (80% accuracy in terms of NMI[12]). It would take many days to cluster this data set using kernel *k*-means and other kernel-based clustering algorithms, while linear clustering algorithms like *k*-means cannot achieve comparable accuracy.

- Batch clustering algorithms such as *k*-means and kernel *k*-means are iterative in nature and need to access the input data points multiple times. However, many data sets are too large

[12]Refer to Section 1.6.2 for the definition of NMI.

to load into the memory, so it would not only be prohibitively expensive to perform multiple passes over the data, but also infeasible to compute the kernel matrix. Some applications such as social network analysis and intrusion detection in networks, involve potentially unbounded sequences of data points called data streams. Only a small subset of the data can be stored, depending on the size of the data buffer. Due to this, each data point can be accessed at most once. This data also evolves over time, so the data points that arrived recently have higher relevance than the older data. There have been relatively few efforts to apply kernel based clustering to data streams, due to the cost of computing the kernel. In Chapter 4, we present an efficient algorithm called *approximate stream kernel k-means*, to perform kernel clustering on stream data. The key idea is to construct the kernel matrix dynamically using importance sampling, and assign labels to the incoming data points in real-time. We use several benchmark data sets to simulate stream data sets, and evaluate the performance of the proposed algorithm on these data sets. We demonstrate that our algorithm is able to cluster stream data sets in real-time with speeds up to 8 MBps.

- Document and image data sets, contain millions of high-dimensional points and usually belong to a large number of categories. Finding clusters in such data sets is computationally expensive using kernel-based clustering techniques because they have quadratic running time complexity in terms of the number of data points, and linear time complexity in terms of the number of dimensions and the number of clusters. Although the approximate kernel clustering algorithms discussed in Chapters 2-4 reduce the running time complexity in terms of the number of data points, their clustering time grows linearly with the number of clusters. In Chapter 5, we present the *sparse kernel k-means algorithm* which can efficiently cluster large data sets into thousands of clusters with significantly lower processing and memory requirements, with high clustering accuracy. It assumes that the kernel matrix is sparse when the number of clusters is large, and constructs a sparse kernel matrix for a subset of the data set, sampled incrementally using importance sampling. Cluster labels are obtained by clus-

Table 1.5 Description of data sets used for evaluation of the proposed algorithms.

Data set	Number of data points n	Dimensionality d	Number of clusters C
CIFAR-10 [99]	60,000	384	10
CIFAR-100 [99]	60,000	384	100
MNIST [108]	70,000	784	10
Forest Cover Type [23]	581,012	54	7
Imagenet-34 [49]	949,401	900	34
Imagenet-164 [49]	1,262,102	900	164
Poker [33]	1,025,010	30	10
Network Intrusion [167]	4,897,988	50	10
Youtube	10,143,254	6,647	N/A
Tiny [173]	79,302,017	384	N/A
Twitter	1,000,000,000	8,042	N/A
Concentric circles	100 to 1,000,000,000	10 to 1,000	10 to 1,000

tering this sparse kernel matrix in a low dimensional space spanned by its top eigenvectors. This algorithm has running time complexity linear in the size and the dimensionality of the data set, and logarithmic in the number of clusters.

1.6 Data sets and Evaluation Metrics

1.6.1 Data sets

To demonstrate the effectiveness of the proposed algorithms, we use several benchmark data sets of different sizes and dimensionalities, from several domains. The description of the data sets is summarized in Table 1.5.

- **MNIST [108]:** The MNIST data set is a subset of the database of handwritten digits available from NIST. It contains $70,000$ images from 10 classes, each class representing one of the digits, $0-9$. Each image is represented as a 784-dimensional feature vector containing the pixel intensity values.
- **Forest Cover Type [23]:** This data set is composed of cartographic variables obtained from

the US Geological Survey (USGS) and the US Forest Service (USFS) data. Each of the $581,012$ data points represents the attributes of a 30×30 square meter cell of the forest floor. There are a total of 12 attributes, including qualitative measures like soil type and wilderness area, and quantitative measures like slope, elevation, and distance to hydrology. These 12 attributes are represented using 54 features. The data are grouped into 7 classes, each representing a different forest cover type. The true cover type was determined from the USFS Region 2 Resource Information System (RIS) data.

- **Imagenet [49]:** The Imagenet data set contains about 14 million images organized according to the Wordnet hierarchy [64]. Each node in this hierarchy represents a concept known as a "synset". We downloaded $1,262,102$ images from $1,000$ synsets, merged the leaf nodes in the synset tree based on their similarity to form a 164-class data set. We call this data set Imagenet-164 and use it to demonstrate the effectiveness of the sparse kernel *k*-means algorithm in Chapter 5. By filtering out the classes with fewer than 500 images, we formed a balanced data set containing $949,401$ images from 34 classes, which we call Imagenet-34 data set. This data set is used to evaluate the remaining clustering algorithms. We computed the Scale Invariant Feature Transform (SIFT) descriptors [117] of the images using the VLFeat library [178], and clustered a randomly chosen subset of 10 million SIFT features to form a visual vocabulary. Each SIFT descriptor was then quantized into a visual word using the nearest cluster center. We obtained a 900-dimensional vector representation for each image, which was then normalized to lie in the range $[0, 1]$.
- **Poker [33]:** This data set, available in the UCI repository [13], contains $1,025,010$ data points. Each data point is an example of a "hand" consisting of five playing cards drawn from a standard deck of 52. Each card is described using two attributes: suit and rank. These attributes are represented using a 30-dimensional categorical feature vector. There are 10 classes in the data set, each depicting a type of poker hand.
- **Network Intrusion [167]:** The Network Intrusion data set contains $4,898,431$ 50-

dimensional data points representing the TCP dump data from seven weeks of a local-area network traffic. The data is classified into 23 classes, one class representing legitimate traffic and the remaining 22 classes representing different types of illegitimate traffic. We filtered out the data from classes which contain fewer than 500 data points, to form a data set with $4,897,988$ data points from 10 classes.

- **Youtube**[13]**:** Youtube is a video hosting website which allows users to upload, view and share videos over the web. It has over one billion users uploading over 300 hours of videos every minute, on a wide range of topics. We used the Youtube Search API[14] to download the meta-data corresponding to $10,143,254$ videos using $26,000$ non-abstract nouns from Wordnet [64] as search queries. We used the video title, description and the video thumbnail (which usually contains the key frame in the video) to extract features for each record. 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 [125]. Feature value $\mathbf{x}_{r,t}$, representing the weight assigned to the term t in record r, measures how important the term is to the record in the data set. It is defined as

$$\mathbf{x}_{r,t} = \text{tf}(r,t) * \text{idf}(t,\mathcal{D}) \tag{1.14}$$

$$= \begin{cases} \frac{1+\log f(r,t)}{\log n - \log f(t)} & \text{if } f(r,t) > 0 \\ 0 & \text{otherwise} \end{cases}, \tag{1.15}$$

where $f(r,t)$ represents the number of times the term t occurs in the record r and $f(t)$ represents the number of records containing the term t. We then downloaded the thumbnail of the video and extracted the global GIST features [141] of the image. The final $6,647$-dimensional feature vector was obtained by concatenating the tf-idf and GIST features. We

[13]www.youtube.com
[14]https://developers.google.com/youtube/v3

use this data set to evaluate the performance of the sparse kernel *k*-means algorithm proposed in Chapter 5 on large high dimensional data sets.

- **Tiny, CIFAR-10 and CIFAR-100 [99, 173]:** The Tiny Image data set contains $79,302,017$ unique 32×32 color images, downloaded from the Internet. They were obtained by extracting $75,062$ non-abstract English nouns from the Wordnet database [64] and using them to search for images in 7 independent image search engines. These images were downloaded and down-sampled to 32×32. We represented each image using a 384-dimensional GIST descriptor [141]. Though the search queries can be used to loosely label the images, these labels are unreliable. To evaluate the accuracy of the proposed algorithms, we used the CIFAR-10 and CIFAR-100 data sets, manually labeled subsets of the Tiny data set. The CIFAR-10 data set contains $60,000$ images from 10 classes (bird, truck, deer, dog, cat, frog, car, plane, horse and ship). The CIFAR-100 also contains $60,000$ images from 100 classes.
- **Twitter**[15]**:** Twitter is a social network with over 100 million active users posting over $100,000$ short messages (called *tweets*) per minute. The tweets contain personal updates, real-time information about events, news etc. Each tweet contains a text message limited to 140 characters and can include user-mentions, links, emoticons, and hashtags in addition to plain text. We downloaded over a billion tweets using the Twitter streaming search API using 20 programming languages (Python, Perl, C#, Java, Ruby, C++, JavaScript, VB-Script, Scala, Objective C, PHP, SQL, Postgresql, GO, Julia, Erlang, HTML, XML, Swift, and ASP.NET) as search terms. We filtered out the non-English tweets, removed the hashtags, eliminated the stop words and represented each tweet with the tf-idf features, defined in (1.15), corresponding to $8,042$ terms. We use this data set to demonstrate the efficiency of the approximate stream kernel *k*-means algorithm in Chapter 4 on fast streaming data sets.

In addition to the above real-world data sets, we use a synthetic data set, which we call the **concentric circles** data set, to demonstrate the scalability of the proposed algorithms. The data set

[15] www.twitter.com

containing circular clusters of varying radii, was generated with different number of points, ranging from 100 to 1 billion. The data dimensionality ranges from 10 to $1,000$ and the number of clusters ranges from 10 to $1,000$. Each cluster contains the same number of points. An example data set containing $1,000$ two-dimensional points along 10 concentric circles (100 points in each cluster) is shown in Figure 4.2(a).

1.6.2 Evaluation Metrics

The goal of our research is to reduce the resources needed for kernel clustering, with minimal reduction in the cluster quality. In order to evaluate the reduction in running time and memory complexity, we measured the time taken for clustering the data points, and the amount of memory used.

The cluster quality of the proposed algorithms were evaluated using two types of measures: (a) internal measures evaluate the structure and compactness of the clusters, while (b) external measures evaluate how well the cluster labels match with the true class labels. We used the internal *Silhouette coefficient* [151] and the external *Normalized Mutual Information (NMI)* [104] measures to evaluate the cluster quality of our algorithms.

The Silhouette coefficient measures the compactness of the clusters. Let $\mathbf{d}_{k,i}$ represent the average dissimilarity between data point $\mathbf{x}_i$ and all the points assigned to the cluster $\mathcal{C}_k$, i.e.

$$\mathbf{d}_{k,i} = \frac{1}{n_k} \sum_{\substack{\mathbf{x}_j \in \mathcal{C}_k \\ \mathbf{x}_i \neq \mathbf{x}_j}} \mathbf{d}^2(\mathbf{x}_i, \mathbf{x}_j),$$

where n_k is the number of points (except for $\mathbf{x}_i$) assigned to cluster $\mathcal{C}_k$. For each data point $\mathbf{x}_i$, define the coefficients a_i and b_i as follows:

$$a_i = \mathbf{d}_{k^*,i}, \text{ and } b_i = \min_{k \neq k^*} \mathbf{d}_{k,i},$$

where k^* is the index of the cluster to which $\mathbf{x}_i$ is assigned. The coefficient a_i represents the average dissimilarity of $\mathbf{x}_i$ with all other points within the same cluster, and the coefficient b_i represents the average dissimilarity between $\mathbf{x}_i$ and all the points in neighboring cluster. The Silhouette coefficient is defined as

$$\text{Silhouette} = \frac{1}{n}\sum_{i=1}^{n}\frac{b_i - a_i}{\max(a_i, b_i)}. \tag{1.16}$$

The value of the Silhouette coefficient lies in the range $[-1, 1]$. A value close to 1 is desired. When the coefficient is close to 1 it implies that $a_i \ll b_i$ for a large number of points, i.e. many of the points are well-matched to the cluster to which they were assigned. On the other hand, when the Silhouette coefficient value is close to -1, $a_i \gg b_i$ for a large number of data points, which implies that many of the points are more similar to the neighboring clusters than the cluster to which they have been assigned. A value close to 0 denotes that many data points lie on the boundaries of their natural clusters.

The Normalized Mutual Information with respect to the true class labels of the data points is defined as follows: Let U^a and U^b be the cluster membership matrices corresponding to two partitions a and b of the same data set. Let n_i^a represent the number of data points that have been assigned label i in partition a, and $n_{i,j}^{a,b}$ represents the number of data points that have been assigned label i in partition a and label j in partition b. We have

$$NMI(a,b) = \frac{\sum_{i=1}^{C}\sum_{j=1}^{C} n_{i,j}^{a,b}\log\left(n\frac{n_{i,j}^{a,b}}{n_i^a n_j^b}\right)}{\sqrt{\left(\sum_{i=1}^{C} n_i^a \log\frac{n_i^a}{n}\right)\left(\sum_{j=1}^{C} n_j^b \log\frac{n_j^b}{n}\right)}}, \tag{1.17}$$

where a represents the partition obtained from the clustering algorithm, and b represents the partition based on the true classes. An NMI value of 1 indicates perfect matching with the true class distribution whereas 0 indicates perfect mismatch. The true class labels are available for most of the data sets (except the Tiny image and Youtube data sets). We used the CIFAR-10 data set, a

labeled subset of the Tiny image data set, to evaluate the performance on the Tiny data set.

1.7 Thesis Overview

Kernel-based clustering algorithms, which perform well on real-world data sets, are not scalable to big data sets, containing billions of high-dimensional points from thousands of clusters. We propose scalable approximate kernel-based clustering algorithms, and demonstrate their efficiency and effectiveness on several diverse large-scale data sets. The remainder of this thesis is organized as follows: Chapters 2 and 3 describe the approximate batch clustering algorithms (approximate kernel *k*-means, and kernel-based clustering using random Fourier features), based on work published in [40] and [42], respectively. These algorithms can cluster up to 10 million data points with thousands of features, and achieve high cluster quality. Chapter 4, based on the publication [43], describes the approximate stream kernel *k*-means algorithm, which can cluster streaming data of arbitrary sizes in real-time. The sparse kernel *k*-means algorithm, discussed in Chapter 5, can cluster arbitrarily-sized high-dimensional data sets, into thousands of clusters. It is applicable to large document and image repositories. This work was published in [38]. We conclude our study and present directions for future work in Chapter 6.

Chapter 2

Approximate Kernel-based Clustering

2.1 Introduction

As discussed in Chapter 1, kernel *k*-means achieves better clustering performance than *k*-means, because it explores the non-linear structure in the data using complex non-linear similarity measures. However, it has running time and memory complexity quadratic in the number of data points n, leading to its non-scalability to big data sets.

To address this issue, we propose an approximate kernel clustering algorithm called *Approximate Kernel k-means* [40], based on random sampling. We sample m points from the data set of n points, and express the cluster centers as linear combinations of vectors in the space spanned by this subset. The weights of the sampled points in the cluster centers, and the cluster labels of the points are obtained simultaneously using iterative optimization. Only a small $n \times m$ portion of the kernel matrix needs to be computed using the proposed algorithm, thereby reducing the running time complexity of clustering to $O(nm)$. When n is in the order of millions, the sample size m is much smaller than n. Hence the proposed algorithm is comparable to *k*-means in terms of efficiency. We show analytically and empirically that the cluster quality achieved by the proposed approximate kernel *k*-means is comparable to that of kernel *k*-means.

This chapter is organized as follows: In Section 2.2, we briefly review some of the popular approximate kernel-based clustering schemes developed in the literature. We formally describe the proposed approximate kernel *k*-means algorithm in Section 2.3. The key parameters which determine the success of the proposed algorithm are the number of samples m and the sampling strategy. We discuss these issues in Section 2.3.1. In Section 2.3.2, we analyze the proposed algorithm's running time and memory complexity. We also show that the difference between the performance of the approximate kernel *k*-means and the kernel *k*-means algorithms in terms of the clustering error[1] reduces as the number of samples m increases, at the rate of $O(1/m)$. In 2.3.3, we present the distributed approximate kernel *k*-means algorithm [41], which parallelizes the proposed approximate kernel *k*-means algorithm, in order to scale up to data sets containing billions of data points. Finally, in Section 2.4, we demonstrate empirically that the proposed approximate clustering algorithm is an efficient and accurate variant of the kernel *k*-means algorithm, and can be used to cluster large data sets, containing billions of points.

2.2 Related Work

Large matrices like the kernel matrices corresponding to large data sets have fast decaying eigenspectrums [187]. Therefore, the computational requirements of operations involving such matrices can be reduced by replacing them with their low-rank approximations. Most of the scalable kernel-based learning algorithms, including the proposed approximate kernel *k*-means algorithm, take advantage of this fact in their design.

Below, we first briefly review the low-rank matrix approximation literature, and then describe some of the large-scale kernel-based clustering algorithms developed in the literature.

[1]Clustering error is defined as the sum of the squared distances between the data points and the center of the cluster to which the data point is assigned. See Section 1.3.1 for the formal definition of clustering error.

2.2.1 Low-rank Matrix Approximation

Given an $n \times m$ matrix A, the objective of low-rank approximation is to find a rank-r matrix A_r that minimizes the error defined by

$$||A - A_r||_p \,,$$

where $||\cdot||_p$ represents either the spectral norm or the Frobenius norm. The optimal solution to (2.2.1) is given by

$$A_r^* = \sum_{k=1}^{r} \lambda_k \mathbf{u}_k \mathbf{v}_k^\top,$$

where $\{\lambda_k\}_{k=1}^r$ represent the largest r singular values of A, and $\{\mathbf{u}_k\}_{k=1}^r$ and $\{\mathbf{v}_k\}_{k=1}^r$ are the corresponding left and right singular vectors [58]. The time required to estimate the singular vectors is $O(mn \min\{m, n\})$, which can be prohibitive when m and n are large.

Several efficient algorithms have been proposed in the literature to approximate the Singular Value Decomposition [129]. One of the earliest algorithms by Frieze *et al.* involves independently sampling s rows and columns from A, to form an $s \times s$ matrix S. A is then projected onto the span of the dominant eigenvectors of S. They showed that when the columns and rows are sampled with probability proportional to the column and row norms respectively, and the sample size $s = O(\max\{r^4\epsilon^{-3}, r^2\epsilon^{-4}\})$, the approximation error can be bounded, with high probability, as

$$||A - A_r||_F^2 = ||A - A_r^*||_F^2 + \epsilon \, ||A||_F^2 \,, \tag{2.1}$$

where $\epsilon > 0$ is an error parameter [70]. Achlioptas *et al.* obtained a similar result by designing a random matrix R with entries dependent on the values in the matrix A, such that the matrix $A + R$ is sparse, and the expectation $\mathbb{E}[R] = 0$. The top r singular vectors of the sparse matrix $A + R$ are used in place of the singular vectors of A to find the low rank approximation of A [5]. The singular vectors of a sparse matrix can be computed efficiently using the Lanczos bidiagonalization method and its variants [163].

Approximation schemes developed thereafter achieved tighter bounds on the error and the sampling complexity, by using different sparsification, sampling and projection schemes [50, 54, 57, 103, 109, 137, 155, 187]. For instance, in [137], matrix A is sparsified by nullifying the entries which have sufficiently low magnitudes. Elements are retained in proportion to their magnitude. Sequential column and row sampling with probability proportional to the column's (row's) distance from the span of the columns (rows) already selected is used in [50]. In [155], A is first projected into a low-dimensional space using a uniform random matrix $R \in \{+1, -1\}^{m \times s}$ as $B = AR$, and then projected onto the span of the best rank-r approximation of B. These works obtained multiplicative error bounds of the form

$$||A - A_r||_p = (1 + \epsilon) \left(||A - A_r^*||_p \right). \tag{2.2}$$

The most widely-studied sampling-based approximation techniques in the literature are the CUR and the Nystrom approximations:

2.2.1.1 CUR matrix approximation

The CUR matrix decomposition method factorizes A as $A \simeq CUR$ where C contains s columns and R contains t rows selected from A, such that $s \ll m, t \ll n$. The $s \times t$ matrix U is constructed to achieve minimal approximation error [22, 44, 55, 122, 184]. Berry *et al.* obtained C are R using Quasi-Gram-Schmidt orthogonalization of A and $A^\top$, respectively [22]. Drineas *et al.* proposed efficient linear-time algorithms which randomly sample the columns and rows, and obtain U through the singular value decomposition of a small $s \times s$ matrix [55]. In [122], they improved the approximation by sampling C and R based on the importance of the columns and rows, measured in terms of their statistical leverage scores[2]. Wang *et al.* augmented a sparsifica-

[2]The statistical leverage score of the i^{th} column of a rank-r $n \times m$ matrix A with singular value decomposition $A = U\Sigma V^\top$ is defined as $\pi_i = \frac{1}{r} \left|\left| V^{(i)} \right|\right|_2^2$, where $V^{(i)}$ represents the i^{th} row in V. It is a measure of the independence of the column and its influence on the matrix.

tion procedure to the CUR decomposition algorithm to further improve its efficiency [184]. The CUR decomposition is preferred over the SVD decomposition in many applications because it can be interpreted more easily.

2.2.1.2 Nystrom matrix approximation

The Nystrom approximation can be viewed as a specialization of the CUR decomposition for symmetric positive semi-definite (SPSD) matrices[3] like kernel similarity matrices [17,57,103,109,111, 170, 187, 195]. It was first used in [187] to perform classification and regression using Gaussian processes. It was then adopted in many kernel-based learning tasks such as classification [187], regression [46, 187], clustering [35, 67, 113], manifold learning [194] and dimensionality reduction [7].

Let K represent an $n \times n$ SPSD matrix and K_r^* represent its best rank-r approximation. The Nystrom approximation studied by Williams *et al.* in [187] samples $m \ll n$ columns uniformly without replacement from K, to form the $n \times m$ matrix K_B. Let $\widehat{K}$ be the $m \times m$ intersection between the sampled columns and the corresponding rows. K is approximated by

$$\widetilde{K} = K_B \widehat{K}^{-1} K_B^\top. \tag{2.3}$$

Drineas *et al.* developed a variant which uses $\widehat{K}_r^*$, the best rank-r approximation of $\widehat{K}$, in place of $\widehat{K}$ in (2.3), and samples the columns uniformly *with* replacement, to obtain approximation error bounds of the form (2.1) [57]. Several non-uniform sampling techniques have been explored in [17, 74, 103, 170, 195] to obtain improved error bounds.

[3] An $n \times n$ matrix K is positive semi-definite if $\mathbf{x}^\top K \mathbf{x} \geq 0$, for all non-zero $\mathbf{x} \in \Re^n$.

2.2.2 Kernel-based Clustering for Large Data sets

Sampling and sparsification techniques have been employed to develop efficient kernel-based clustering algorithms.

The spectral clustering algorithm is a graph-based clustering technique [118]. The n points in the data set are represented as nodes of a graph. Each edge in the graph is weighted by the similarity between the points connected by the edge. Let K denote the $n \times n$ similarity matrix. The spectral clustering algorithm uses the first C eigenvectors of the Laplacian matrix defined by

$$L = I - \text{diag}(K^\top \mathbf{1})^{-1/2} K \text{diag}(K^\top \mathbf{1})^{-1/2},$$

to find the clusters. The obvious computational bottlenecks in this algorithm are the calculation of the Laplacian matrix and the computation of its eigenvectors, which require $O(n^2)$ and $O(n^3)$ time respectively.

The column sampling method by Frieze *et al.* and Nystrom approximation method have been used in the literature to speed up spectral clustering [18, 67, 102]. The key idea is to approximate the Laplacian matrix, and use the approximate eigenvectors to find the clusters. The running time complexity of these approximate spectral clustering algorithms is $O(nm + m^3)$, where m is the number of columns sampled from the kernel matrix. As m is usually much lower than n, these algorithms run much faster than spectral clustering. Random projection can be combined with sampling to further improve the clustering efficiency [73, 153]. A low-dimensional projection of the similarity matrix, obtained by multiplying it with a random Gaussian matrix, is used to construct the graph Laplacian matrix. These approximate spectral clustering algorithms have been applied successfully in image segmentation problems [113].

Nystrom approximation was also used to accelerate the kernel neural gas algorithm [145]. The objective of kernel neural gas is to find C prototypes to represent the data. Each prototype is associated with a randomly initialized weight, which is updated by a factor proportional to the

similarity between the prototype and the input data point. Similar to the kernel *k*-means, the prototypes are expressed as linear combinations of the data points in the Hilbert space, so each weight update involves the $n \times n$ kernel matrix. The Nystrom approximation of the kernel matrix was used in [156] to perform the weight updates, thereby reducing its running time complexity.

In addition to the above approximate methods, several heuristic and application-specific algorithms have been proposed to perform efficient kernel-based clustering. Zhang and Rudnicky reduced the memory requirements of the kernel *k*-means algorithm by changing the order in which clustering is performed. The kernel matrix is computed blockwise, and the cluster labels are obtained by examining only one block at a time [196]. The KASP algorithm first clusters the input data set into m clusters using *k*-means, and then executes spectral clustering on the m ($C \ll m \ll n$) cluster centers to obtain the C clusters [191]. The RASP clustering method first partitions the data space using Random Projection (RP) trees [191]. RP trees are data structures that partition the data space into m cells, by splitting recursively along one randomly chosen coordinate at a time. Each cell in the partition is represented by its center, and spectral clustering is executed on the m representative centers. These methods reduce the running time complexity of clustering to $O(nm + m^3)$. Chen *et al.* sparsified the similarity matrix by retaining only the similarity values corresponding to the nearest p neighbors for each node, and proposed a simple scheme to parallelize the similarity computation and clustering [35]. The nearest neighbors are found using *kd*-trees [131] and metric trees [176], thereby reducing the overall memory requirement to $O(np)$, although the running time complexity is still $O(n^2 \log(p))$. The GEM (Graph Extraction + weighted kernel *k*-Means) algorithm proposed in [186] speeds up kernel *k*-means for social network graphs by eliminating the nodes with low degree. It makes use of the power law distribution of social networks, which indicates that a small set of high degree vertices cover a large portion of the network.

2.3 Approximate Kernel k-means

Given a data set $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, and a kernel function $\kappa(\cdot, \cdot)$, kernel *k*-means finds C clusters, whose centers $\mathbf{c}_k(\cdot)$ are represented as linear combinations of all the points in the data set, in accordance with the representer theorem [158], i.e.

$$\mathbf{c}_k(\cdot) = \sum_{i=1}^{n} \widehat{U}_{k,i} \kappa(\mathbf{x}_i, \cdot), k \in [C], \tag{2.4}$$

where $\widehat{U}$ is the cluster membership matrix normalized by the number of points in each cluster, as defined in (2.8). In other words, the cluster centers lie in the subspace spanned by all the data points, i.e. $\mathbf{c}_k(\cdot) \in \mathcal{H}_\kappa = \text{span}(\kappa(\mathbf{x}_1, \cdot), \ldots, \kappa(\mathbf{x}_n, \cdot)), k \in [C]$. As a consequence, the kernel *k*-means algorithm requires the computation of $O(n^2)$ kernel similarity values, leading to its non-scalability.

We can avoid computing the full kernel matrix if we restrict the solution for the cluster centers to a smaller subspace $\mathcal{H}_a \subset \mathcal{H}_\kappa$. $\mathcal{H}_a$ should be constructed such that

(i) $\mathcal{H}_a$ is small enough to allow efficient computation, and

(ii) $\mathcal{H}_a$ is rich enough to yield data partitions similar to those obtained using $\mathcal{H}_\kappa$.

We employ a simple randomized approach for constructing $\mathcal{H}_a$: we randomly sample m data points ($m \ll n$), denoted by $\widehat{\mathcal{D}} = \{\widehat{\mathbf{x}}_1, \ldots, \widehat{\mathbf{x}}_m\}$, and construct the subspace $\mathcal{H}_a = \text{span}(\widehat{\mathbf{x}}_1, \ldots, \widehat{\mathbf{x}}_m)$. Given the subspace $\mathcal{H}_a$, we modify the kernel *k*-means optimization problem (1.7) as

$$\min_{U \in \mathcal{P}} \max_{\{\mathbf{c}_k(\cdot) \in \mathcal{H}_a\}_{k=1}^{C}} \sum_{k=1}^{C} \sum_{i=1}^{n} U_{k,i} \left|\left|\mathbf{c}_k(\cdot) - \kappa(\mathbf{x}_i, \cdot)\right|\right|_{\mathcal{H}_\kappa}^2, \tag{2.5}$$

where $U = (\mathbf{u}_1, \ldots, \mathbf{u}_C)^\top$ is the cluster membership matrix, $\mathbf{c}_k(\cdot) \in \mathcal{H}_a, k \in [C]$ are the cluster centers, and domain $\mathcal{P} = \{U \in \{0,1\}^{C \times n} : U^\top \mathbf{1} = \mathbf{1}\}$, where $\mathbf{1}$ is a vector of all ones. Let $K_B \in \Re^{n \times m}$ represent the kernel similarity matrix between data points in $\mathcal{D}$ and the sampled data

points $\widehat{\mathcal{D}}$, and $\widehat{K} \in \Re^{m \times m}$ represent the kernel similarity between the sampled data points. The following lemma allows us to reduce (2.5) to an optimization problem involving only the cluster membership matrix U.

Lemma 1. *Given the cluster membership matrix U, the optimal cluster centers in* (2.5) *are given by*

$$\mathbf{c}_k(\cdot) = \sum_{i=1}^{m} \alpha_{k,i} \kappa(\widehat{\mathbf{x}}_i, \cdot), \tag{2.6}$$

where $\alpha = \widehat{U} K_B \widehat{K}^{-1}$. The optimization problem for U is given by

$$\min_{U} \textit{tr}(K) - \textit{tr}(\widetilde{U} K_B \widehat{K}^{-1} K_B^{\top} \widetilde{U}^{\top}), \tag{2.7}$$

where $\widehat{U}$ and $\widetilde{U}$ are defined by

$$\begin{aligned} \widehat{U} &= (\widehat{\mathbf{u}}_1, \ldots, \widehat{\mathbf{u}}_C)^{\top} = [\textit{diag}(n_1, \ldots, n_C)]^{-1} U, \\ \widetilde{U} &= (\widetilde{\mathbf{u}}_1, \ldots, \widetilde{\mathbf{u}}_C)^{\top} = [\textit{diag}(\sqrt{n_1}, \ldots, \sqrt{n_C})]^{-1} U, \textit{ and} \\ n_k &= \mathbf{u}_k^{\top} \mathbf{1}, k \in [C]. \end{aligned} \tag{2.8}$$

Proof. Let $\varphi_i = (\kappa(\mathbf{x}_i, \widehat{\mathbf{x}}_1), \ldots, \kappa(\mathbf{x}_i, \widehat{\mathbf{x}}_m))$ and $\alpha_i = (\alpha_{i,1}, \ldots, \alpha_{i,m})$ be the i^{th} rows of matrices K_B and α respectively. As $\mathbf{c}_k(\cdot) \in \mathcal{H}_a = \text{span}(\widehat{\mathbf{x}}_1, \ldots, \widehat{\mathbf{x}}_m)$, we can express $\mathbf{c}_k(\cdot)$ as

$$\mathbf{c}_k(\cdot) = \sum_{i=1}^{m} \alpha_{k,i} \kappa(\widehat{\mathbf{x}}_i, \cdot),$$

and write the objective function in (2.5) as

$$\begin{aligned}
&\sum_{k=1}^{C}\sum_{i=1}^{n} U_{k,i}\,||\mathbf{c}_k(\cdot)-\kappa(\mathbf{x}_i,\cdot)||^2_{\mathcal{H}_\kappa} \\
&\sum_{k=1}^{C}\sum_{i=1}^{n} U_{k,i}\left|\left|\sum_{j=1}^{m}\alpha_{k,j}\kappa(\widehat{\mathbf{x}}_j,\cdot)-\kappa(\mathbf{x}_i,\cdot)\right|\right|^2_{\mathcal{H}_\kappa} \\
= \quad & \mathrm{tr}(K)+\sum_{k=1}^{C}\left(n_k\alpha_k^\top\widehat{K}\alpha_k-2\mathbf{u}_k^\top K_B\alpha_k\right). \qquad (2.9)
\end{aligned}$$

By minimizing the above expression with respect to α_k, we have

$$\alpha_k=\widehat{K}^{-1}K_B^\top\widehat{\mathbf{u}}_k, k\in[C] \qquad (2.10)$$

and therefore, $\alpha=\widehat{U}K_B\widehat{K}^{-1}$. We complete the proof by substituting the expression for α into (2.9). □

As indicated by Lemma 1, we need to compute only K_B for finding the cluster memberships. $\widehat{K}$ is part of K_B and therefore does not need to be computed separately. When $m \ll n$, this computational cost would be significantly smaller than that of computing the full matrix.

We refer to the proposed algorithm as **Approximate Kernel *k*-means**, outlined in Algorithm 3. Figure 2.1 illustrates the algorithm on a two-dimensional synthetic data set containing two semi-circles. Except for a few points which are misclustered, the result is similar to that of kernel *k*-means. Table 2.1 compares the confusion matrices of the partitions obtained using the approximate kernel *k*-means algorithm, with those of the kernel *k*-means and the *k*-means algorithms. A confusion matrix shows the mapping between the true class labels and the cluster labels. Each cluster is assigned a class label, corresponding to the true label of the majority of the data points in the cluster. Each entry (k, c) in the confusion matrix represent the number of data points from class c assigned to cluster k. The diagonal entries represent the number of points that have been assigned to the correct cluster. It is clear from Table 2.1 that the proposed algorithm achieves cluster quality

comparable to that of the kernel *k*-means algorithm, and is much more accurate than the *k*-means algorithm.

Algorithm 3 Approximate Kernel *k*-means

Input:
- $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}, \mathbf{x}_i \in \Re^d$: the set of n d-dimensional data points to be clustered
- $\kappa(\cdot, \cdot) : \Re^d \times \Re^d \mapsto \Re$: the kernel function
- C: the number of clusters
- m: the number of randomly sampled data points ($C < m \ll n$)
- $MAXITER$: maximum number of iterations

Output: Cluster membership matrix $U \in \{0, 1\}^{C \times n}$

Sample m data points from $\mathcal{D}$, denoted by $\widehat{\mathcal{D}} = \{\widehat{\mathbf{x}}_1, \ldots, \widehat{\mathbf{x}}_m\}$.

Compute matrices $K_B = [\kappa(\mathbf{x}_i, \widehat{\mathbf{x}}_j)]_{n \times m}$, $\widehat{K} = [\kappa(\widehat{\mathbf{x}}_i, \widehat{\mathbf{x}}_j)]_{m \times m}$, and $T = K_B \widehat{K}^{-1}$.

Randomly initialize the membership matrix U, ensuring that $U^\top \mathbf{1} = \mathbf{1}$.

Set $t = 0$.

repeat

  Set $t = t + 1$.

  Compute the ℓ_1 normalized membership matrix $\widehat{U}$ by $\widehat{U} = [\text{diag}(U\mathbf{1})]^{-1} U$.

  Calculate $\alpha = \widehat{U} T$.

  for $i = 1, \ldots, n$ **do**

    Find the closest cluster center k_* for $\mathbf{x}_i$ by

$$k_* = \underset{k \in [C]}{\operatorname{argmin}} \, \alpha_k^\top \widehat{K} \alpha_k - 2\varphi_i^\top \alpha_k,$$

    where α_k and φ_i are the k^{th} and i^{th} rows of matrices α and K_B, respectively.

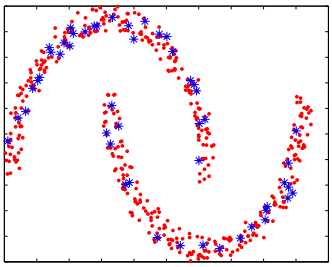
    Update the i^{th} column of U by $U_{k,i} = 1$ for $k = k_*$ and zero otherwise.

  end for

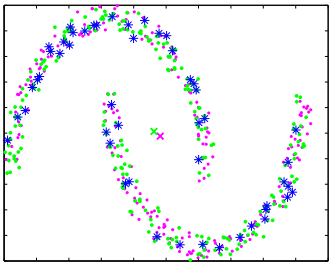
until the membership matrix U does not change, or $t > MAXITER$

2.3.1 Parameters

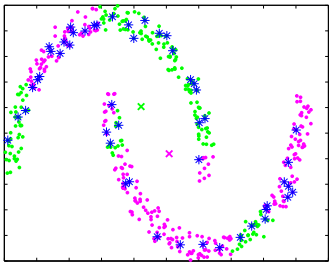
In addition to the kernel function and the number of clusters, the approximate kernel *k*-means is parameterized by the sample size m and the random sampling technique employed to obtain the subset $\widehat{\mathcal{D}}$. These parameters play a crucial role in determining the clustering performance of the proposed algorithm.



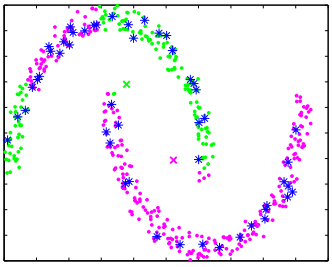
(a)



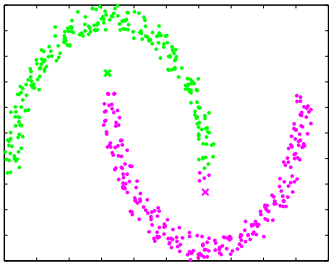
(b)



(c)



(d)



(e)

Figure 2.1 Illustration of the approximate kernel *k*-means algorithm on the two-dimensional semi-circles data set containing 500 points (250 points in each of the two clusters). Figure (a) shows all the data points (in red) and the uniformly sampled points (in blue). Figures (b)-(e) show the process of discovery of the two clusters in the data set and their centers in the input space (represented by x) by the approximate kernel *k*-means algorithm.

Table 2.1 Comparison of the confusion matrices of the approximate kernel *k*-means, kernel *k*-means and *k*-means algorithms for the two-dimensional semi-circles data set, containing 500 points (250 points in each of the two clusters). The approximate kernel *k*-means algorithm achieves cluster quality comparable to that of the kernel *k*-means algorithm.

	Class 1	**Class 2**
Cluster 1	245	4
Cluster 2	5	246

(a) Approximate kernel *k*-means

	Class 1	**Class 2**
Cluster 1	250	0
Cluster 2	0	250

(b) Kernel *k*-means

	Class 1	**Class 2**
Cluster 1	132	129
Cluster 2	118	121

(c) *k*-means

2.3.1.1 Sample size

By comparing the optimization problem of approximate kernel *k*-means in (2.7) with the kernel *k*-means problem in (1.10), we can observe that the approximate kernel *k*-means problem can be viewed as the kernel *k*-means problem in which the kernel matrix K is replaced by its Nystrom approximation $K_B\widehat{K}^{-1}K_B^\top$. Therefore, the clustering performance of the approximate *k*-means problem will be close to the clustering performance of kernel *k*-means if the approximation error $\left|\left|K - K_B\widehat{K}^{-1}K_B^\top\right|\right|$ is small. The following lemma adapted from [74] characterizes the number of samples required to obtain a good approximation.

Lemma 2. *Let $\{\lambda_k, \mathbf{v}_k\}_{k=1}^n$ denote the eigenvalues and eigenvectors of the kernel matrix K. Let $V_C = (\mathbf{v}_1, \ldots, \mathbf{v}_C)$ denote the eigenvectors corresponding to the dominant C eigenvalues of K. Define the "coherence" of the dominant C-dimensional invariant subspace of K as*

$$\tau = \frac{n}{C}\max_{1\leq i\leq n}\left|\left|V_C^{(i)}\right|\right|_2^2, \tag{2.11}$$

where $V_C^{(i)}$ is the i^{th} row in V_C. Assume that the eigengap $\lambda_C - \lambda_{C+1}$ is sufficiently large. For any $\delta \in (0,1)$, we have

$$\left|\left|K - K_B\widehat{K}^{-1}K_B^\top\right|\right|_2 \leq \lambda_{C+1}\left(1 + \frac{2n}{m}\right),$$

with probability $1-\delta$, provided $m \geq 8\tau C\log(C/\delta)$.

The coherence of a matrix τ is a measure of the number of informative columns in the matrix. When the coherence is low, few columns are sufficient to obtain an accurate approximation. Lemma 2 indicates that the approximation error reduces at a rate of $O(1/m)$, with increasing m.

In our experiments, we examined the performance of our algorithm for different sample sizes m, ranging from 0.001% to 15% of the data set size n, and observed that setting m equal to 0.01% to 0.05% of n leads to a satisfactory performance.

2.3.1.2 Sampling strategies

Another important factor that influences the proposed approximate kernel *k*-means algorithm is the sampling distribution employed to construct the kernel approximation. The simplest sampling technique is uniform random sampling, i.e. each point is selected with a probability $1/n$. Several non-uniform sampling and greedy approaches to perform low-rank matrix approximation, have been studied in the literature.

(i) Diagonal sampling involves choosing a data point $\mathbf{x}_i$ with a probability proportional to the diagonal element $K(\mathbf{x}_i, \mathbf{x}_i)$ [17, 57]. This distribution is the same as the uniform distribution for exponential kernels of the form

$$\kappa(\mathbf{x}_a, \mathbf{x}_b) = \exp\left(-\lambda \left|\left|\mathbf{x}_a - \mathbf{x}_b\right|\right|_p^q\right), p, q > 0,$$

such as the RBF kernel and the Laplacian kernel, because all the diagonal entries are equal to one another.

(ii) Column-norm sampling involves choosing x_i with a probability proportional to the ℓ_2 norm of the column vector $K(\cdot, \mathbf{x}_i)$ [69].

(iii) In [195], *k*-means is applied to the data set and the cluster centers obtained are used in place of the sampled data set $\widehat{\mathcal{D}}$.

(iv) Adaptive sampling techniques involve selecting data points sequentially, to ensure maximum coverage of the data [50, 102, 114, 142]. For example, a greedy selection procedure which selects a point which is farthest from the currently selected set of points is employed in [142]. Liu *et al.* propose selecting a data point which would form a subspace with the previously chosen points, so that the total distance of unsampled data points to this subspace is minimized [114].

(v) Sampling based on the importance of the data point in terms of the statistical leverage scores and the coherence of the data is employed in [170].

The non-uniform sampling techniques like column-norm sampling, adaptive sampling and importance sampling have $O(n^2)$ running time complexity. Hence, they are infeasible for large data sets. Sampling using *k*-means can be performed in $O(nm)$ time. Uniform and diagonal sampling have linear time complexity. Kumar *et al.* compared the diagonal and column sampling techniques with uniform sampling and showed that uniform sampling without replacement is more effective than the non-uniform sampling techniques [103]. We explore some of these techniques empirically in Section 2.4.

2.3.2 Analysis

In this section, we first analyze the computational complexity of the proposed approximate kernel *k*-means algorithm, and then examine the quality of the data partitions generated by the proposed algorithm.

2.3.2.1 Computational complexity

Assuming uniform sampling strategy, sampling can be performed in $O(n)$ time. The most expensive operations in the proposed algorithm are the matrix inversion $\widehat{K}^{-1}$ and calculation of the matrix $T = K_B\widehat{K}^{-1}$, which have a total computational cost of $O(m^3 + m^2n)$. The cost of computing α and updating the membership matrix U is $O(mnCl)$, where l is the number of iterations needed for convergence. Hence, the overall running time complexity of the approximate kernel *k*-means algorithm is $O(m^3+m^2n+mnCl)$. We can further reduce the computational complexity by avoiding the matrix inversion $\widehat{K}^{-1}$ and formulating the calculation of $\alpha = \widehat{U}T = \widehat{U}K_B\widehat{K}^{-1}$ as

the following optimization problem:

$$\min_{\alpha \in \Re^{C \times m}} \frac{1}{2}\text{tr}(\alpha \widehat{K} \alpha) - \text{tr}(\widehat{U} K_B \alpha^\top) \tag{2.12}$$

If $\widehat{K}$ is well conditioned (i.e. the minimum eigenvalue of $\widehat{K}$ is significantly larger than zero), we can solve the optimization problem in (2.12) by a simple gradient descent method with a convergence rate of $O\left(\log(1/\varepsilon)\right)$, where ε is the desired accuracy. As the computational cost of each step in the gradient descent method is $O(m^2 C)$, the overall computational cost is only $O(m^2 Cl \log(1/\varepsilon)) \ll O(m^3)$ when $Cl \ll m$. This reduces the overall computational cost to $O(m^2 Cl + mnCl + m^2 n)$. As the largest matrix that needs to be stored in memory is K_B, the memory requirement is only $O(mn)$. This is a dramatic decrease in the running time and memory requirements for large data sets when compared to the $O(n^2)$ complexity of kernel *k*-means. The running time complexity of approximate kernel *k*-means is also lower than that of the Nystrom approximation based spectral clustering algorithm, which needs to compute the eigenvectors of the $m \times m$ matrix $\widehat{K}$ in $O(m^3)$ time.

2.3.2.2 Approximation error

In this section, we compare the clustering error of approximate kernel *k*-means with that of kernel *k*-means. The only difference between the two algorithms is the fact that approximate kernel *k*-means restricts the cluster centers to a small subspace $\mathcal{H}_a$, constructed using the sampled data points. Our analysis will therefore be focused on bounding the expected error due to this constraint.

Let binary random variables $\xi = (\xi_1, \xi_2, ..., \xi_n)^\top \in \{0, 1\}^n$ represent the sampling vector, i.e. $\xi_i = 1$ if $\mathbf{x}_i \in \widehat{\mathcal{D}}$ and zero otherwise. The following proposition allows us to write the clustering error in terms of random variable ξ:

Proposition 1. *Given the cluster membership matrix* $U = (\mathbf{u}_1, \ldots, \mathbf{u}_C)^\top$*, the clustering error can*

be expressed in ξ *as*

$$\mathcal{L}(U,\xi) = tr(K) + \sum_{k=1}^{C} \mathcal{L}_k(U,\xi), \tag{2.13}$$

where $\mathcal{L}_k(U,\xi)$ *is*

$$\mathcal{L}_k(U,\xi) = \min_{\alpha_k \in \Re^n} -2\mathbf{u}_k^\top K(\alpha_k \circ \xi) + n_k(\alpha_k \circ \xi)^\top K(\alpha_k \circ \xi). \tag{2.14}$$

Note that approximate kernel *k*-means becomes equivalent to kernel *k*-means when $\xi = \mathbf{1}$, where $\mathbf{1}$ is a vector of all ones, implying that all the data points are selected for constructing the subspace $\mathcal{H}_a$. As a result, $\mathcal{L}(U,\mathbf{1})$ is the clustering error of the kernel *k*-means algorithm.

The following lemma relates the expected clustering error of approximate kernel *k*-means with that of kernel *k*-means.

Lemma 3. *Given the membership matrix* U*, we have the expectation of* $\mathcal{L}(U,\xi)$ *bounded as follows*

$$\mathbb{E}_\xi[\mathcal{L}(U,\xi)] \leq \mathcal{L}(U,\mathbf{1}) + tr\left(\widetilde{U}\left[K^{-1} + \frac{m}{n}[diag(K)]^{-1}\right]^{-1}\widetilde{U}^\top\right), \tag{2.15}$$

where $\mathcal{L}(U,\mathbf{1}) = tr(K) - tr(\widetilde{U}K\widetilde{U}^\top)$.

Proof. We first bound $\mathbb{E}_\xi[\mathcal{L}_k(U,\xi)]$ as

$$\begin{aligned}
& \frac{1}{n_k}\mathbb{E}_\xi[\mathcal{L}_k(U,\xi)] \\
= \;& \mathbb{E}_\xi\left[\min_\alpha -2\widehat{\mathbf{u}}_k^\top K(\alpha\circ\xi) + (\alpha\circ\xi)^\top K(\alpha\circ\xi)\right] \\
\leq \;& \min_\alpha \mathbb{E}_\xi\left[-2\widehat{\mathbf{u}}_k^\top K(\alpha\circ\xi) + (\alpha\circ\xi)^\top K(\alpha\circ\xi)\right] \\
= \;& \min_\alpha -2\frac{m}{n}\widehat{\mathbf{u}}_k^\top K\alpha + \frac{m^2}{n^2}\alpha^\top K\alpha + \frac{m}{n}\left(1-\frac{m}{n}\right)\alpha^\top \mathrm{diag}(K)\alpha \\
\leq \;& \min_\alpha -2\frac{m}{n}\widehat{\mathbf{u}}_k^\top K\alpha + \frac{m}{n}\alpha^\top\left(\frac{m}{n}K + \mathrm{diag}(K)\right)\alpha.
\end{aligned}$$

By minimizing the above expression with respect to α, we obtain

$$\alpha^* = \left(\frac{m}{n}K + \text{diag}(K)\right)^{-1} K\widehat{\mathbf{u}}_k.$$

Therefore,

$$\frac{1}{n_k}\mathbb{E}_\xi[\mathcal{L}_k(U,\xi)] \leq -\frac{m}{n}\widehat{\mathbf{u}}_k K \left(\frac{m}{n}K + \text{diag}(K)\right)^{-1} K\widehat{\mathbf{u}}_k.$$

$\mathbb{E}_\xi[\mathcal{L}_k(U,\xi)]$ can be bounded as

$$\begin{aligned} & \mathbb{E}_\xi[\mathcal{L}_k(U,\xi)] + n_k\widehat{\mathbf{u}}_k^\top K\widehat{\mathbf{u}}_k \\ \leq\ & n_k\widehat{\mathbf{u}}_k^\top \left(K - K\left[K + \frac{n}{m}\text{diag}(K)\right]^{-1} K\right)\widehat{\mathbf{u}}_k \\ =\ & \widetilde{\mathbf{u}}_k^\top \left(K^{-1} + \frac{m}{n}[\text{diag}(K)]^{-1}\right)^{-1}\widetilde{\mathbf{u}}_k. \end{aligned}$$

We complete the proof by adding up $\mathbb{E}_\xi[\mathcal{L}_k(U,\xi)]$ and using the fact that

$$\mathcal{L}_k(U,\mathbf{1}) = \min_\alpha -2\mathbf{u}_k^\top K\alpha + n_k\alpha^\top K\alpha = -\widetilde{\mathbf{u}}_k^\top K\widetilde{\mathbf{u}}_k.$$

□

The above result can be interpreted in terms of the eigenvalues of the kernel matrix.

Corollary 1. *Assume $\kappa(\mathbf{x},\mathbf{x}) \leq 1$ for any $\mathbf{x}$. Let $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n \geq 0$ be the eigenvalues of matrix K. Given the membership matrix U, we have*

$$\begin{aligned} \frac{\mathbb{E}_\xi[\mathcal{L}(U,\xi)]}{\mathcal{L}(U,\mathbf{1})} &\leq 1 + \frac{\sum_{i=1}^C \lambda_i/[1+\lambda_i m/n]}{tr(K) - \sum_{i=1}^C \lambda_i} \\ &\leq 1 + \frac{C/m}{\sum_{i=C+1}^n \lambda_i/n}. \end{aligned}$$

Proof. As $\kappa(\mathbf{x},\mathbf{x}) \leq 1$ for any $\mathbf{x}$, we have $\text{diag}(K) \preceq I$, where I is an identity matrix. As $\widetilde{U}$ is an

ℓ_2 normalized matrix, we have

$$\begin{aligned} & \text{tr}\left(\widetilde{U}\left[K^{-1}+\frac{m}{n}[\text{diag}(K)]^{-1}\right]^{-1}\widetilde{U}^{\top}\right) \\ \leq\ & \text{tr}\left(\widetilde{U}\left[K^{-1}+\frac{m}{n}I\right]^{-1}\widetilde{U}^{\top}\right) \\ \leq\ & \sum_{i=1}^{C}\frac{\lambda_i}{1+m\lambda_i/n}\leq\frac{Cn}{m} \end{aligned}$$

and

$$\mathcal{L}(U,\mathbf{1})=\text{tr}(K-UKU^{\top})\geq\text{tr}(K)-\sum_{i=1}^{C}\lambda_i.$$

We complete the proof by combining the above inequalities. □

To illustrate the result of Corollary 1, consider a special kernel matrix K that has its first a eigenvalues equal n/a and the remaining eigenvalues equal zero; i.e. $\lambda_1=\ldots=\lambda_a=n/a$ and $\lambda_{a+1}=\ldots=\lambda_n=0$. We further assume $a>2C$; i.e. the number of non-zero eigenvalues of K is larger than twice the number of clusters. Then, according to Corollary 1, we have

$$\frac{\mathbb{E}_{\xi}[\mathcal{L}(U,\xi)]-\mathcal{L}(U,\mathbf{1})}{\mathcal{L}(U,\mathbf{1})}\leq 1+\frac{Ca}{m(a-C)}\leq 1+\frac{2C}{m},$$

indicating that when the number of non-zero eigenvalues of K is significantly larger than the number of the clusters, the difference in the clustering errors of kernel *k*-means and our approximation scheme will decrease at the rate of $O(1/m)$. This result concurs with the result of Lemma 1.

2.3.3 Distributed Clustering

As the proposed approximate kernel *k*-means algorithm has $O(nm)$ running time complexity, it is easier to parallelize than the kernel *k*-means algorithm. In Algorithm 4, we propose a scheme to parallelize approximate kernel *k*-means. The key idea is to distribute the kernel computation and perform approximate clustering using a relatively smaller matrix.

Algorithm 4 Distributed Approximate Kernel *k*-means

Input:
- $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}, \mathbf{x}_i \in \Re^d$: the set of n d-dimensional data points to be clustered
- $\kappa(\cdot,\cdot) : \chi \times \chi \mapsto \Re$: the kernel function
- C: the number of clusters
- m: the number of randomly sampled data points ($C < m \ll n$)
- P: the number of tasks
- $MAXITER$: maximum number of iterations

Output: Cluster membership matrix $U \in \{0,1\}^{C \times n}$

// Master task

Randomly sample m data points from $\mathcal{D}$, denoted by $\widehat{\mathcal{D}} = \{\widehat{\mathbf{x}}_1, \ldots, \widehat{\mathbf{x}}_m\}$ and compute $\widehat{K} = [\kappa(\widehat{\mathbf{x}}_i, \widehat{\mathbf{x}}_j)]_{m \times m}$.

Randomly split the unsampled data points into P parts $\left\{\mathcal{D}^1, \ldots, \mathcal{D}^P\right\}$.

Execute in parallel:

// Task *l*

Compute $K_B^l = [\kappa(\mathbf{x}_i, \widehat{\mathbf{x}}_j)]_{s \times m}$ and $T^l = K_B^l \widehat{K}^{-1}$, where $\mathbf{x}_i \in \mathcal{D}^l$ and s is the number of points in $\mathcal{D}^l$.

Randomly initialize the membership matrix U^l, ensuring that ${U^l}^\top \mathbf{1} = \mathbf{1}$.

Set $t = 0$.

repeat

  Set $t = t + 1$.

  Calculate $\alpha^l = [\text{diag}(U^l \mathbf{1})]^{-1} U^l T^l$.

  for $i = 1, \ldots, s$ **do**

    Find the closest cluster center k_* for $\mathbf{x}_i \in \mathcal{D}^l$ by

$$k_* = \underset{k \in [C]}{\operatorname{argmin}} \left(\alpha_k^l\right)^\top \widehat{K} \left(\alpha_k^l\right) - 2 \left(\varphi_i^l\right)^\top \left(\alpha_k^l\right),$$

    where α_k^l and φ_i^l are the k^{th} and i^{th} rows of matrices α^l and K_B^l, respectively.

    Update the i^{th} column of U^l by $U_{k,i}^l = 1$ for $k = k_*$ and zero otherwise.

  end for

until the membership matrix U does not change or $t > MAXITER$

for each point $\mathbf{x}_i \notin \mathcal{D}^l$ **do**

  Find the closest cluster center k_*

$$k_* = \underset{k \in [C]}{\operatorname{argmin}} \left(\alpha_k^l\right)^\top \widehat{K} \left(\alpha_k^l\right) - 2 \left(\varphi_i^l\right)^\top \left(\alpha_k^l\right),$$

  where α_k^l is the k^{th} row in α^l and $\varphi_i^l = (\kappa(\mathbf{x}_i, \widehat{\mathbf{x}}_1), \ldots, \kappa(\mathbf{x}_i, \widehat{\mathbf{x}}_m))$.

  Update the i^{th} column of U^l by $U_{k,i}^l = 1$ for $k = k_*$ and zero otherwise.

end for

end parallel execution

// Master task

Randomly select an index l and set $U = U^l$, or combine the matrices $\left\{U^l\right\}_{l=1}^P$ using an ensemble clustering algorithm (e.g. the Meta-clustering algorithm described in Algorithm 5).

Algorithm 5 Meta-Clustering Algorithm

Input: Cluster membership matrices $\left\{U^l\right\}_{l=1}^{P}, U^l \in \{0,1\}^{C\times n}$

Output: Consensus cluster membership matrix U

Concatenate the membership matrices $\{U^l\}_{l=1}^{P}$ to obtain an $PC \times n$ matrix $\mathcal{U} = (\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_{PC})^\top$.

Compute the Jaccard similarity $s_{i,j}$ between the vectors $\mathbf{u}_i$ and $\mathbf{u}_j$, $i, j \in [PC]$ using

$$s_{i,j} = \frac{\mathbf{u}_i^\top \mathbf{u}_j}{\|\mathbf{u}_i\|^2 + \|\mathbf{u}_j\|^2 - \mathbf{u}_i^\top \mathbf{u}_j}.$$

Construct a complete weighted meta-graph $G = (V, E)$, where vertex set $V = \{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_{PC}\}$ and each edge $(\mathbf{u}_i, \mathbf{u}_j)$ is weighted by $s_{i,j}$.

Partition G into C meta-clusters $\{\pi_k\}_{k=1}^{C}$ where $\pi_k = \left\{\mathbf{u}_k^{(1)}, \mathbf{u}_k^{(2)}, \ldots \mathbf{u}_k^{(s_k)}\right\}$.

Compute the mean vectors for each meta-cluster: $\{\mu_k\}_{k=1}^{C}$ using

$$\mu_k = \frac{1}{s_k} \sum_{i=1}^{s_k} \mathbf{u}_k^{(i)}.$$

for $i = 1, \ldots, n$ **do**

    Update the i^{th} column of U as

$$U_{k_*,i} = \begin{cases} 1 & \text{if } k_* = \underset{k\in[C]}{\arg\max}\, \mu_{k,i} \\ 0 & \text{otherwise} \end{cases}$$

end for

We first sample m points $\widehat{\mathcal{D}} = \{\widehat{\mathbf{x}}_1, \ldots, \widehat{\mathbf{x}}_m\}$ from the data set and randomly split the remaining $n - m$ data points into P parts $\left\{\mathcal{D}^1, \ldots, \mathcal{D}^P\right\}$. Let the matrix $\widehat{K} = \kappa(\widehat{\mathbf{x}}_i, \widehat{\mathbf{x}}_j)$ where $\widehat{\mathbf{x}}_i, \widehat{\mathbf{x}}_j \in \widehat{\mathcal{D}}$. We then map each partition to a processing node. Each node computes the kernel matrix $K_B^l = \kappa(\mathbf{x}_i, \widehat{\mathbf{x}}_j)$, where $\mathbf{x}_i \in \mathcal{D}^l$, the set of points assigned to the node, and finds the cluster labels for the s points in $\mathcal{D}^l$ and the corresponding cluster centers, using the matrices K_B^l and $\widehat{K}$. Each point $\mathbf{x}_i \notin \mathcal{D}^l$ is assigned to the cluster whose center is closest. This process generates P cluster membership matrices $\left\{U^l\right\}_{l=1}^{P}, U^l \in \{0,1\}^{C\times n}$. To obtain the final cluster membership matrix U, we can either randomly choose one index l and set $U = U^l$, or combine them using an ensemble clustering algorithm.

The objective of ensemble clustering [180] is to combine multiple partitions of the given data set. A popular ensemble clustering algorithm is the Meta-Clustering algorithm (MCLA) [168], described in Algorithm 5. It maximizes the average normalized mutual information between the partitions using hypergraph partitioning. Given P cluster membership matrices, $\{U^1, \ldots, U^P\}$, where $U^l = (\mathbf{u}_1^l, \ldots, \mathbf{u}_C^l)^\top$, the objective of this algorithm is to find a consensus membership matrix U that maximizes the Average Normalized Mutual Information, defined as

$$ANMI = \frac{1}{P}\sum_{k=1}^{P} NMI(U, U^l), \tag{2.16}$$

where $NMI(U^a, U^b)$, the Normalized Mutual Information (NMI) [104] between two partitions a and b, represented by the membership matrices U^a and U^b respectively, is defined by

$$NMI(U^a, U^b) = \frac{\sum_{i=1}^{C}\sum_{j=1}^{C} n_{i,j}^{a,b} \log\left(n\frac{n_{i,j}^{a,b}}{n_i^a n_j^b}\right)}{\sqrt{\left(\sum_{i=1}^{C} n_i^a \log\frac{n_i^a}{n}\right)\left(\sum_{j=1}^{C} n_j^b \log\frac{n_j^b}{n}\right)}}. \tag{2.17}$$

In equation (2.17), n_i^a represents the number of data points that have been assigned label i in partition a, and $n_{i,j}^{a,b}$ represents the number of data points that have been assigned label i in partition a and label j in partition b. NMI values lie in the range $[0, 1]$. An NMI value of 1 indicates perfect matching between the two partitions whereas 0 indicates perfect mismatch. Maximizing (2.16) is a combinatorial optimization problem and solving it exhaustively is computationally infeasible. MCLA obtains an approximate consensus solution by representing the set of partitions as a hypergraph. Each vector $\mathbf{u}_k^l, k \in [C], l \in [P]$ represents a vertex in a regular undirected graph, called the *meta-graph*. Vertex $\mathbf{u}_i$ is connected to vertex $\mathbf{u}_j$ by an edge whose weight is proportional to the Jaccard similarity between the two vectors $\mathbf{u}_i$ and $\mathbf{u}_j$:

$$s_{i,j} = \frac{\mathbf{u}_i^\top \mathbf{u}_j}{\|\mathbf{u}_i\|^2 + \|\mathbf{u}_j\|^2 - \mathbf{u}_i^\top \mathbf{u}_j}. \tag{2.18}$$

This meta-graph is partitioned using a graph partitioning algorithm such as METIS [93] to obtain C balanced meta-clusters $\{\pi_1, \pi_2, \ldots \pi_C\}$. Each meta-cluster $\pi_k = \left\{\mathbf{u}_k^{(1)}, \mathbf{u}_k^{(2)}, \ldots \mathbf{u}_k^{(s_k)}\right\}$, containing s_k vertices, is represented by the mean vector

$$\mu_k = \frac{1}{s_k} \sum_{i=1}^{s_k} \mathbf{u}_k^{(i)}. \tag{2.19}$$

The value $\mu_{k,i}$ represents the association between data point $\mathbf{x}_i$ and the k^{th} cluster. Each data point $\mathbf{x}_i$ is assigned to the meta-cluster with which it is associated the most, breaking ties randomly, i.e

$$U_{k_*,i} = \begin{cases} 1 & \text{if } k_* = \underset{k \in [C]}{\arg\max}\, \mu_{k,i} \\ 0 & \text{otherwise} \end{cases} \tag{2.20}$$

By parallelizing the approximate kernel *k*-means algorithm, the running time complexity for kernel calculation and clustering reduces to $O(nm/P)$ and $O(m^2C + mnC/P + m^2n/P)$, respectively. If the ensemble clustering algorithm is employed to combine the partitions in the last step, an additional cost of $O(nC^2P^2)$ is incurred. The communication overhead is minimal. Only the m sampled data points need to be replicated, in contrast to the n number of data points that need to be replicated across all the nodes in parallel kernel *k*-means.

2.4 Experimental Results

In this section, we show that the approximate kernel *k*-means algorithm is an efficient and scalable variant of the kernel *k*-means algorithm. It has lower running time and memory requirements but is on par with kernel *k*-means in terms of the clustering quality.

2.4.1 Data sets

We use the medium-sized CIFAR-10 and MNIST data sets, for which it is feasible but expensive to compute the $n \times n$ kernel matrix, to demonstrate that the proposed algorithm's clustering performance is similar to that of the kernel *k*-means algorithm, in terms of the cluster quality. We then demonstrate the efficiency of the proposed algorithm on large data sets, on a single processor, using the large Forest Cover Type, Imagenet-34, Poker and Network Intrusion data sets. We analyze the scalability of our algorithm using the synthetic concentric circles data set. We finally execute the distributed approximate kernel *k*-means on the Tiny data set and the concentric circles data set containing a billion points.

2.4.2 Baselines

We first compared the proposed technique with the kernel *k*-means algorithm to show that similar performance is achieved by our algorithm. We also gauged our algorithm's performance against that of the Nystrom spectral clustering algorithm [67], which clusters the top C eigenvectors of a low rank approximate kernel matrix, obtained through the Nystrom approximation technique, and the *k*-means algorithm to show that our algorithm achieves better cluster quality.

2.4.3 Parameters

To define the inter-point similarity, we used the universal RBF kernel with the kernel width parameter set equal to $\rho\mathfrak{d}$, where $\mathfrak{d}$ is the average pairwise Euclidean distance between the data points, and parameter ρ is a value in the range $[0, 1]$[4]. The value which achieved the best NMI was employed. We evaluated the efficiency of the proposed algorithm for different sample sizes ranging from $m = 100$ to $m = 2,000$. We selected these sample sizes to ensure that the true clusters in each data set are sufficiently represented in the sample, with high probability. For the purpose of

[4]The average pairwise similarity was used only as a heuristic to set the RBF kernel width, and not required by the proposed algorithm. Other techniques may be employed to choose the kernel and the kernel parameters.

evaluation, the number of clusters C was set equal to the number of true classes in the data set. All algorithms were implemented in MATLAB[5] and run on a 2.8 GHz processor. The memory used was explicitly limited to 40 GB. We executed each algorithm 10 times and present the results averaged over these runs. Different permutations of the data set were input to the algorithm in each run.

2.4.4 Results

2.4.4.1 Running time

Table 2.2 Running time (in seconds) of the proposed approximate kernel *k*-means and the baseline algorithms. The sample size m is set to $2,000$, for both the proposed algorithm and the Nystrom approximation based spectral clustering algorithm. It is not feasible to execute kernel *k*-means on the large Forest Cover Type, Imagenet-34, Poker, and Network Intrusion data sets due to their large size. An approximate value of the running time of kernel *k*-means on these data sets is obtained by first executing kernel *k*-means on a randomly chosen subset of $50,000$ data points to find the cluster centers, and then assigning the remaining points to the closest cluster center.

Data set	**Approximate kernel *k*-means (proposed)**	**Nystrom approximation based spectral clustering**	**Kernel *k*-means**	***k*-means**
CIFAR-10	37.01 (± 6.52)	116.13 (± 1.97)	725.32 (± 7.39)	159.22 (± 75.81)
MNIST	57.73 (± 12.94)	4,186.02 (± 386.17)	914.59 (± 235.14)	448.69 (± 177.24)
Forest Cover Type	157.48 (± 27.37)	573.55 (± 327.49)	4,721.03 (± 504.21)	40.88 (± 6.4)
Imagenet-34	1,261.02 (± 37.39)	1,841.47 (± 123.82)	154,416.48 ($\pm 32,302.44$)	31,076.41 ($\pm 9,355.41$)
Poker	256.26 (± 44.84)	520.48 (± 51.29)	9,942.40 ($\pm 1,476.00$)	40.88 (± 6.40)
Network Intrusion	891.08 (± 237.17)	1,682.46 (± 235.70)	34,784.56 ($\pm 1,493.59$)	953.41 (± 169.38)

[5]We used the *k*-means implementation in the MATLAB Statistics Toolbox and the Nystrom approximation based spectral clustering implementation [35] available at http://alumni.cs.ucsb.edu/ wychen/sc.html. The remaining algorithms were implemented in-house.

Figure 2.2 Example images from three clusters in the Imagenet-34 data set. The clusters represent (a) butterfly, (b) odometer, and (c) website images.

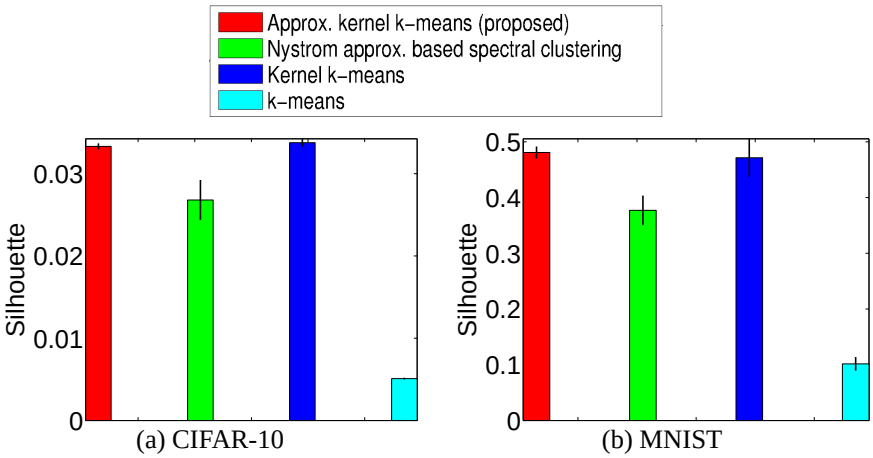
The running times of the proposed algorithm for sample size $m = 2,000$ and the baseline algorithms are recorded in Table 2.2. We observed that a speedup of over 90% was achieved by our algorithm when compared to kernel *k*-means on the CIFAR-10 and MNIST data sets. It is infeasible to calculate the $n \times n$ kernel for the large Forest Cover Type, Imagenet-34, Poker and Network Intrusion data sets. To gauge the efficiency of our algorithm against kernel *k*-means on these data sets, we randomly selected a set of $50,000$ points from these data sets, executed kernel *k*-means on this subset, and assigned cluster labels to the remaining points by finding the cluster whose center is closest. Our algorithm was faster than this version of the kernel *k*-means algorithm as well. Even the *k*-means algorithm was slower than the proposed approximate kernel *k*-means algorithm on most of the data sets, due to their high dimensionality. Our algorithm was also faster than the spectral clustering algorithm based on the Nystrom approximation, because spectral clustering requires the eigendecomposition of the similarity matrix. The most time-consuming operation in our algorithm, computation of the inverse matrix $\widehat{K}^{-1}$, heavily influenced the clustering time.

2.4.4.2 Cluster quality

Figures 2.2 and 2.5 show examples of clusters obtained, using the approximate kernel *k*-means algorithm, from the Imagenet-34 and the CIFAR-10 data sets, respectively. We assigned a class label to each cluster, based on the true class of majority of the objects in the cluster.

The silhouette coefficients of the proposed algorithm are compared with those of the baseline

algorithms, on the CIFAR-10 and MNIST data sets, in Figure 2.3. Computing the silhouette coefficient values for the partitions of the remaining data sets is computationally prohibitive. On both the CIFAR-10 and MNIST data sets, the silhouette coefficient values achieved by the proposed algorithm are close to those of the kernel *k*-means algorithm, proving that the two algorithms yield similar partitions. The Nystrom approximation based spectral clustering algorithm achieves lower silhouette values, while the *k*-means algorithm achieves values close to 0, showing that the clusters obtained are not compact. The NMI values achieved by the proposed algorithm against the base-



(a) CIFAR-10 (b) MNIST

Figure 2.3 Silhouette coefficient values of the partitions obtained using approximate kernel *k*-means, compared to those of the partitions obtained using the baseline algorithms. The sample size m is set to $2,000$, for both the proposed algorithm and the Nystrom approximation based spectral clustering algorithm.

line algorithms are shown in Figure 2.4. Due to the small size of the images in the CIFAR-10 data set, it is difficult to obtain a high clustering accuracy on this data set. Despite this difficulty, our algorithm partitioned the images into clusters similar to those obtained by using kernel *k*-means. The MNIST data set was also clustered into partitions similar to the partitions obtained from kernel *k*-means. Our algorithm's prediction accuracy in terms of NMI with respect to the true class labels is comparable to that of kernel *k*-means. The proposed algorithm's NMI values are marginally better than those of the approximate spectral clustering algorithm, because the spectral clustering algorithm uses only the top C eigenvectors of the kernel matrix to determine the clusters, which

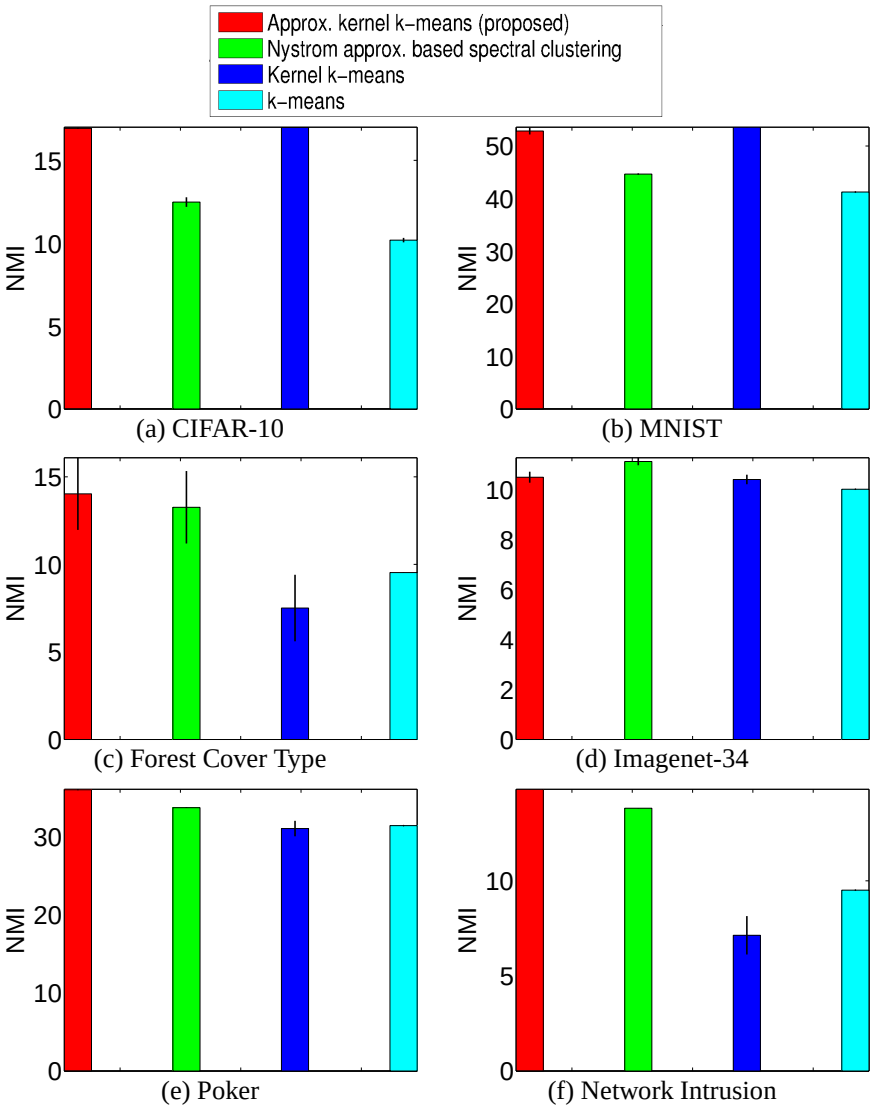


Figure 2.4 NMI values (in %) of the partitions obtained using approximate kernel *k*-means, with respect to the true class labels. The sample size m is set to $2,000$, for both the proposed algorithm and the Nystrom approximation based spectral clustering algorithm. It is not feasible to execute kernel *k*-means on the large Forest Cover Type, Imagenet-34, Poker, and Network Intrusion data sets due to their large size. The approximate NMI values of kernel *k*-means on these data sets are obtained by first executing kernel *k*-means on a randomly chosen subset of $50,000$ data points to find the cluster centers, and then assigning the remaining points to the closest cluster center.

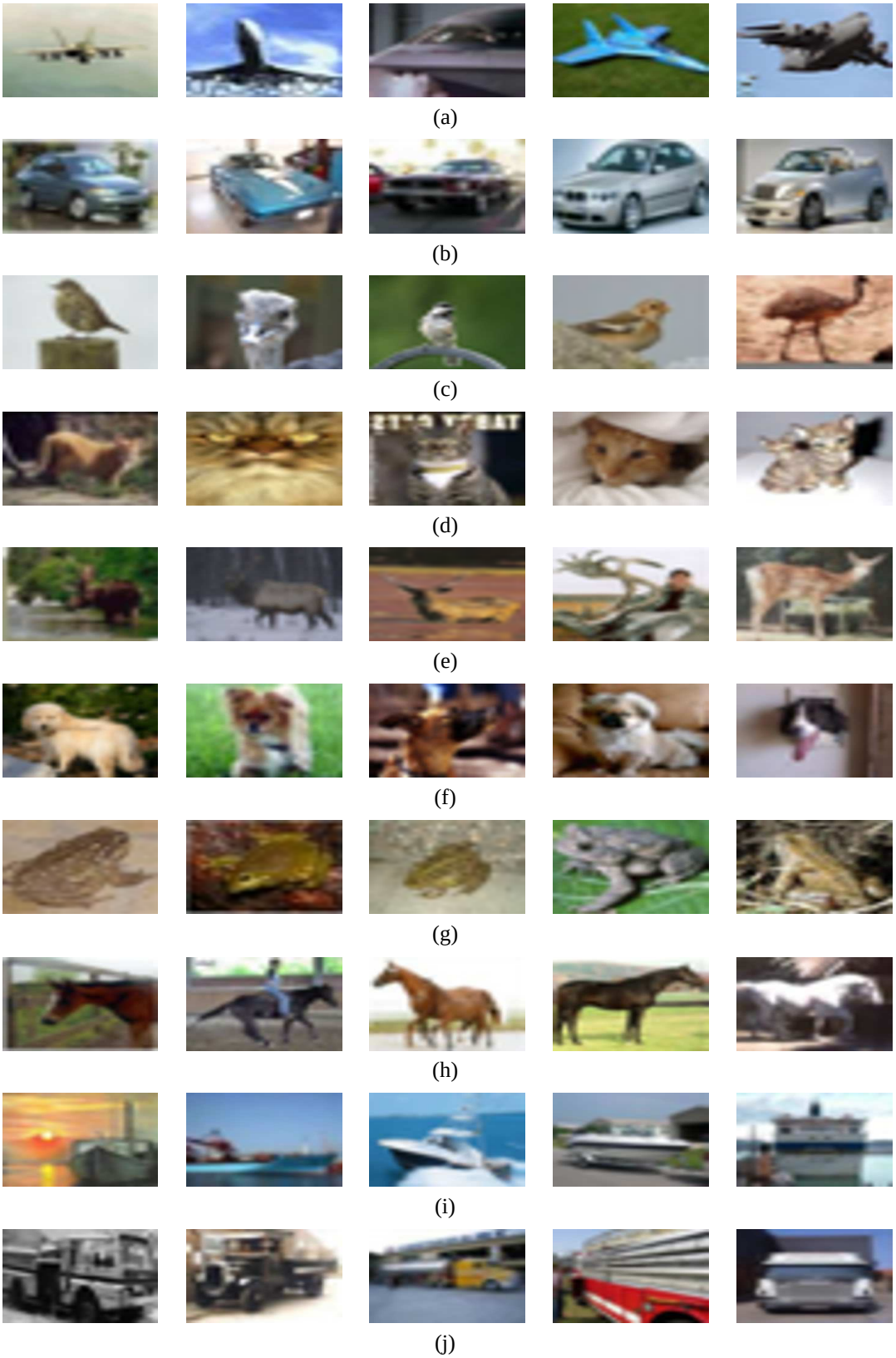


Figure 2.5 Example images from the clusters found in the CIFAR-10 data set using approximate kernel *k*-means. The clusters represent the following objects: (a) airplane, (b) automobile, (c) bird, (d) cat, (e) deer, (f) dog, (g) frog, (h) horse, (i) ship, and (j) truck.

may be too restrictive for these data sets. As expected, all the kernel-based algorithms performed better than *k*-means.

2.4.4.3 Parameter sensitivity

The proposed approximate kernel *k*-means algorithm is dependent on one crucial parameter: the sample size m. We study the effect of varying this parameter on the running time of the algorithm in Table 2.3, and the cluster quality in Figure 2.6 (NMI values) and Figure 2.7 (Silhouette coefficient values). We compare the performance of our algorithm against the Nystrom approximation based spectral clustering algorithm, which also depends on the same parameter. In Table 2.3, the execution time is split into the time taken for computing the kernel matrix and clustering the data points. The kernel computation time is common to the proposed algorithm and the Nystrom approximation based spectral clustering algorithm. More time was spent in clustering than in kernel calculation, due to the simplicity of the RBF kernel. Though our algorithm took longer than the approximate spectral clustering algorithm for small sample sizes ($m \leq 1,000$), the running time of the spectral clustering algorithm increased cubically with the number of samples. Our algorithm was faster for large sample sizes, when high cluster quality was achieved. The running time of our algorithm also increased as the sample size m increased, but at a lower rate. The silhouette coefficient values of the proposed algorithm increased marginally as the sample size increased, and were higher than those achieved by the Nystrom approximation based spectral clustering algorithm. The NMI values achieved by our algorithm were also higher than those achieved by the Nystrom approximation based spectral clustering algorithm, especially when the sample size is large, and spectral clustering is computationally expensive. Only on the Imagenet-34 data set, our algorithm performs marginally worse than the spectral clustering algorithm. There is a marginal improvement in the NMI of our algorithm as the sample size increases.

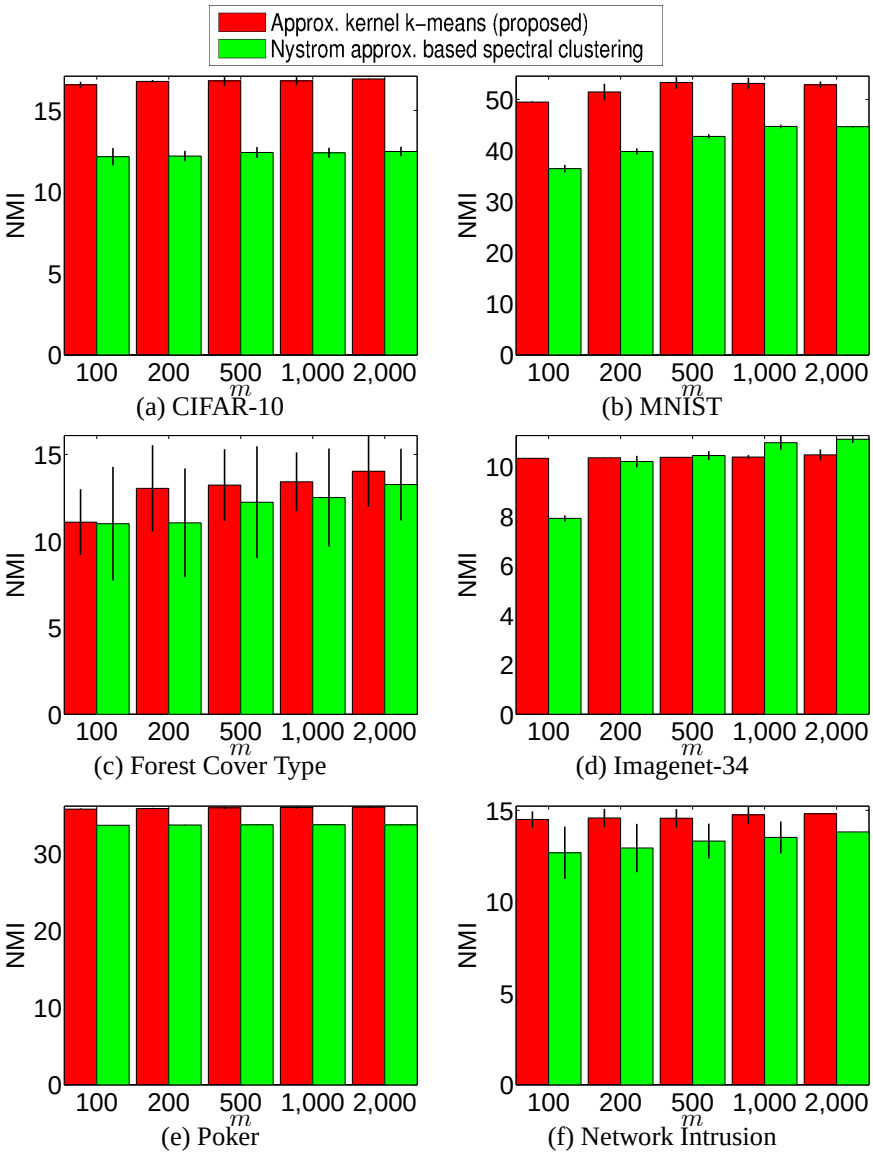
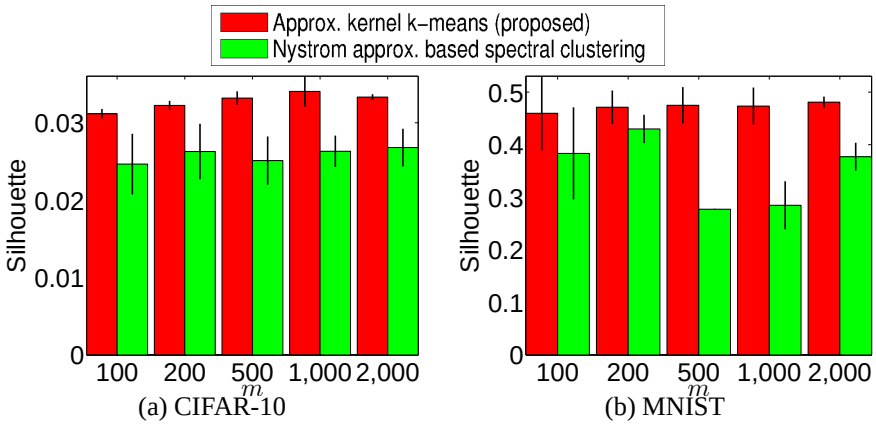


Figure 2.6 Effect of the sample size m on the NMI values (in %) of the partitions obtained using approximate kernel *k*-means, with respect to the true class labels.



(a) CIFAR-10 (b) MNIST

Figure 2.7 Effect of the sample size m on the Silhouette coefficient values of the partitions obtained using approximate kernel *k*-means.

2.4.4.4 Sampling strategies

In our implementation of the proposed algorithm, we employed uniform random sampling to select the subset of data using which the kernel matrix is constructed. Other sampling strategies such as column-norm sampling, diagonal sampling and *k*-means based sampling may be used to select the samples. Table 2.4, Figure 2.8, and Figure 2.9 compare the running time, silhouette coefficient and NMI values, respectively, for the column norm sampling and the *k*-means sampling strategies with uniform random sampling. For column norm-sampling, we assume that the $n \times n$ kernel matrix is pre-computed and only record the time taken for computing the column norms, and the time taken for choosing the first m indices, as the sampling time. For *k*-means sampling, we record the time taken to execute *k*-means and find the representative samples. As expected, the sampling time for both the non-uniform sampling techniques was greater than the time required for uniform random sampling. Column norm sampling is more expensive than *k*-means sampling, after the kernel computation time is taken into account. Both the non-uniform sampling techniques are as accurate as uniform random sampling for substantially large sample sizes, both in terms of silhouette coefficient values as well as the NMI values. This shows that the additional time spent for non-uniform sampling does not lead to significant improvement in the performance, aligning

Table 2.3 Effect of the sample size m on the running time (in seconds) of the proposed approximate kernel *k*-means clustering algorithm.

m	**Approx. kernel calculation**	**Approx. kernel *k*-means (proposed)**	**Nystrom approx. based spectral clustering**
100	0.34 ($\pm$0.04)	11.95 ($\pm$4.62)	0.57 ($\pm$0.12)
200	0.87 ($\pm$0.07)	39.04 ($\pm$15.04)	0.99 ($\pm$0.13)
500	1.36 ($\pm$0.03)	11.84 ($\pm$2.11)	4.25 ($\pm$1.86)
1,000	3.63 ($\pm$0.23)	45.87 ($\pm$21.94)	22.61 ($\pm$5.03)
2,000	4.60 ($\pm$0.20)	32.41 ($\pm$6.32)	111.53 ($\pm$1.77)

(a) CIFAR-10

m	**Approx. kernel calculation**	**Approx. kernel *k*-means (proposed)**	**Nystrom approx. based spectral clustering**
100	0.65 ($\pm$0.06)	25.91 ($\pm$3.05)	7.20 ($\pm$1.00)
200	1.06 ($\pm$0.18)	14.54 ($\pm$7.85)	49.56 ($\pm$9.19)
500	1.99 ($\pm$0.34)	21.36 ($\pm$8.35)	348.86 ($\pm$107.43)
1,000	3.32 ($\pm$0.44)	25.78 ($\pm$6.78)	920.34 ($\pm$219.62)
2,000	5.81 ($\pm$0.35)	51.92 ($\pm$12.59)	4,180.21 ($\pm$385.82)

(b) MNIST

m	**Approx. kernel calculation**	**Approx. kernel *k*-means (proposed)**	**Nystrom approx. based spectral clustering**
100	1.40 ($\pm$0.29)	17.70 ($\pm$6.06)	10.35 ($\pm$1.44)
200	1.64 ($\pm$0.09)	22.57 ($\pm$12.39)	16.83 ($\pm$2.38)
500	3.82 ($\pm$0.03)	28.56 ($\pm$11.61)	50.11 ($\pm$10.83)
1,000	11.14 ($\pm$0.68)	55.01 ($\pm$18.57)	137.26 ($\pm$40.88)
2,000	22.80 ($\pm$1.27)	134.68 ($\pm$26.10)	550.75 ($\pm$326.22)

(c) Forest Cover Type

m	**Approx. kernel calculation**	**Approx. kernel *k*-means (proposed)**	**Nystrom approx. based spectral clustering**
100	47.29 ($\pm$1.12)	504.41 ($\pm$119.41)	78.53 ($\pm$7.14)
200	68.15 ($\pm$0.16)	608.24 ($\pm$10.78)	115.16 ($\pm$4.47)
500	168.83 ($\pm$0.27)	737.24 ($\pm$209.26)	292.69 ($\pm$7.21)
1,000	181.93 ($\pm$11.95)	847.06 ($\pm$22.88)	404.73 ($\pm$79.77)
2,000	344.39 ($\pm$3.77)	916.63 ($\pm$33.62)	1497.08 ($\pm$120.05)

(d) Imagenet-34

Table 2.3 (cont'd)

m	Approx. kernel calculation	Approx. kernel *k*-means (proposed)	Nystrom approx. based spectral clustering
100	2.85 ($\pm$0.36)	53.02 ($\pm$10.86)	10.88 ($\pm$1.65)
200	7.31 ($\pm$1.40)	81.83 ($\pm$30.72)	46.78 ($\pm$4.21)
500	12.74 ($\pm$2.41)	104.83 ($\pm$17.76)	90.57 ($\pm$18.57)
1,000	31.29 ($\pm$2.64)	171.55 ($\pm$41.61)	261.14 ($\pm$20.51)
2,000	40.75 ($\pm$3.83)	215.51 ($\pm$41.01)	479.73 ($\pm$47.46)

(e) Poker

m	Approx. kernel calculation	Approx. kernel *k*-means (proposed)	Nystrom approx. based spectral clustering
100	7.52 ($\pm$0.64)	729.07 ($\pm$237.67)	241.84 ($\pm$65.00)
200	13.82 ($\pm$4.15)	683.22 ($\pm$438.10)	200.48 ($\pm$45.24)
500	41.36 ($\pm$10.75)	339.77 ($\pm$119.48)	436.79 ($\pm$206.47)
1,000	87.24 ($\pm$10.54)	551.39 ($\pm$78.01)	668.91 ($\pm$49.37)
2,000	115.14 ($\pm$7.06)	775.94 ($\pm$230.11)	1567.32 ($\pm$228.64)

(f) Network Intrusion

with the results of earlier works such as [103].

2.4.4.5 Scalability analysis

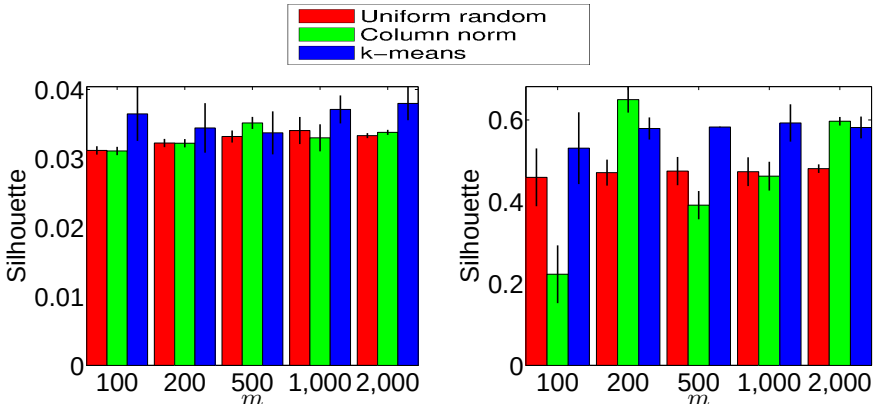
We analyze the scalability of the proposed approximate kernel *k*-means for different values of n, d, C using the synthetic concentric circles data set. We employed the RBF kernel function to compute the approximate kernel matrices, and set the number of sampled points $m = 1,000$ when $C < 100$ and $m = 10C$ when $C \geq 100$. This was done in order to ensure that the condition imposed by Lemma 2 is satisfied.

Figure 2.10(a) shows that the running time of the algorithm varies nearly linearly as the number of points in the data set n varies from 100 to 10 million, the dimensionality $d = 100$, and the number of clusters $C = 10$. This concurs with our complexity analysis in Section 2.3.2.

We set the number of data points $n = 10^6$ and the number of clusters $C = 10$, and studied the effect of the data dimensionality on the performance of the proposed algorithm in Figure 2.10(b). The dimensionality of the data set plays an important role only in the calculation of the kernel. The RBF kernel is simple and takes only a few 100 seconds to calculate, even for $n = 10^6$. The running

Table 2.4 Comparison of sampling times (in milliseconds) of the uniform, column-norm and *k*-means sampling strategies on the CIFAR-10 and MNIST data sets. Parameter m represents the sample size.

m	CIFAR-10			MNIST		
	Uniform Random	**Column norm (×e03)**	***k*-means (×e06)**	**Uniform Random**	**Column norm (×e03)**	***k*-means (×e06)**
100	9.62 (±1.62)	67.62 (±2.31)	1.68 (±0.43)	9.41 (±1.74)	94.22 (±3.97)	3.83 (±0.542)
200	4.24 (±1.12)	68.21 (±3.49)	1.90 (±0.20)	9.34 (±1.16)	88.92 (±4.44)	2.62 (±0.254)
500	3.99 (±0.65)	64.54 (±4.26)	2.14 (±0.14)	11.10 (±3.81)	86.27 (±0.94)	7.82 (±3.42)
1,000	5.43 (±0.87)	67.42 (±5.59)	2.44 (±0.16)	8.41 (±1.38)	86.15 (±0.70)	5.88 (±1.78)
2,000	4.62 (±2.20)	70.43 (±7.20)	2.66 (±0.03)	9.53 (±1.94)	86.66 (±0.85)	4.91 (±0.207)



(a) CIFAR-10 (b) MNIST

Figure 2.8 Comparison of Silhouette coefficient values of the partitions obtained from approximate kernel *k*-means using the uniform, column-norm and *k*-means sampling strategies, on the CIFAR-10 and MNIST data sets. Parameter m represents the sample size.

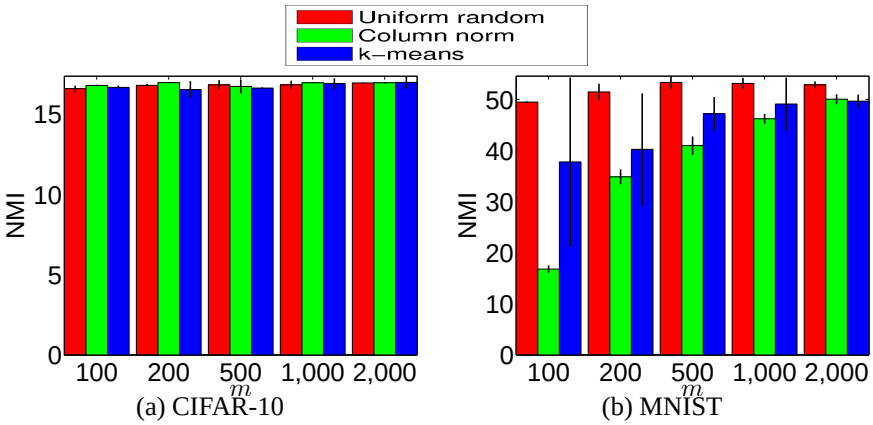


Figure 2.9 Comparison of NMI values (in %) of the partitions obtained from approximate kernel *k*-means using the uniform, column-norm and *k*-means sampling strategies, on the CIFAR-10 and MNIST data sets. Parameter m represents the sample size.

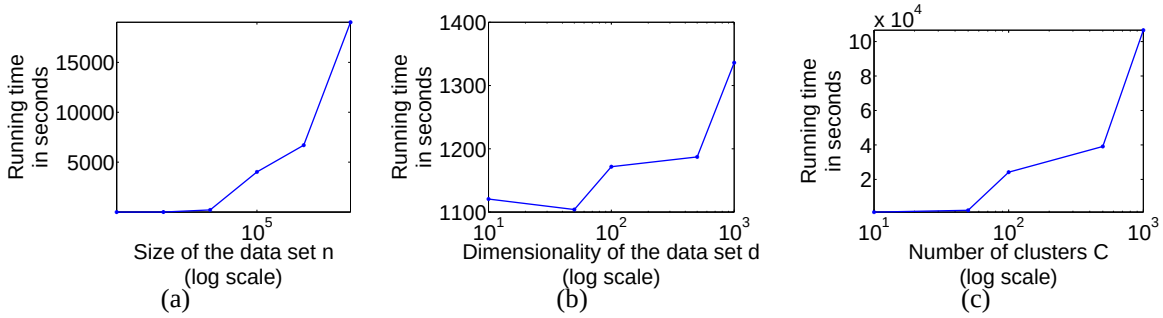


Figure 2.10 Running time of the approximate kernel *k*-means algorithm for different values of (a) n, (b) d and (c) C.

time is dominated by the time taken for clustering. As a result, the running time varies minimally when the dimensionality of the data set varies from $d = 10$ to $d = 1,000$.

We fixed $n = 10^6$ and $d = 100$, increased the number of clusters in the data set from $C = 10$ to $C = 1,000$, and recorded the running time of our algorithm in Figure 2.10(c). As expected, the running time almost increases linearly with C. When $C < 100$, the number of samples m is fixed to $1,000$. Therefore, the number of clusters has a significant effect only on the clustering time. When $C \geq 100$, the number of samples m also need to be increased, thereby affecting both the kernel calculation time and the clustering time.

2.4.5 Distributed Approximate Kernel *k*-means

On data sets of sizes greater than 10 million, execution of approximate kernel *k*-means on a single processor is highly time-consuming. We employed the distributed approximate kernel *k*-means to cluster the Tiny images data set and the synthetic concentric circles data set.

We set the sample size $m = 1,000$ and the number of tasks $P = 1,024$. Each task was run on a 2.8 GHz processor, with a total of 100 GB shared memory. The RBF kernel was used for both data sets. The number of clusters was set to $C = 100$ and $C = 10$ for the Tiny images and concentric circles data sets, respectively.

The clustering performance of the distributed algorithm on the two data sets is presented in Table 2.5. When approximate kernel *k*-means was executed on the Tiny images data set on a single processor, it took about 8.5 hours. The distributed algorithm is able to cluster this data set in under 2 minutes. The concentric circles data set containing 1 billion points was also clustered in less than 15 minutes. The true class labels are not available for the Tiny image data set, so it was not possible to evaluate the cluster quality. On the concentric circles data set, an NMI of about 78% was achieved.

Table 2.5 Performance of the distributed approximate kernel *k*-means algorithm on the Tiny image data set and the concentric circles data set, with parameters $m = 1,000$ and $P = 1024$.

Data set		**Tiny**	**Concentric circles**
n		79,302,017	1,000,000,000
d		384	10
C		100	10
Running time	**Kernel calculation**	0.21 (± 0.07)	1.17 (± 0.09)
	Clustering	94.03 (± 6.58)	876.75 (± 163.06)
NMI		N/A	77.80 (± 0.10)

2.5 Summary

In this chapter, we presented the approximate kernel *k*-means algorithm, an efficient approximation for the kernel *k*-means clustering algorithm, suitable for big data sets. The key to the efficiency of approximate kernel *k*-means is the fact that it does not require the calculation of the pairwise similarities between all the data points. By restricting the cluster centers to lie in a subspace spanned by a small set of randomly sampled data points, it is able to compute the clusters using only a small portion of the kernel matrix. Consequently, it has lower running time and memory complexity than kernel *k*-means and other kernel-based clustering algorithms. We have shown theoretically that, the difference in the clustering error of the approximate kernel *k*-means and the kernel *k*-means algorithms, reduces linearly as the number of sampled points increases. Experimental results also show that the performance of approximate kernel *k*-means is comparable to that of kernel *k*-means and other state-of-the-art approximate kernel clustering algorithms, in terms of the cluster quality, while its running time is close to that of linear clustering algorithms such as *k*-means. Though not as easily parallelizable as *k*-means, it requires lesser data replication and communication than kernel *k*-means. Hence, it can handle distributed data sets more efficiently than kernel *k*-means. The proposed approximate kernel *k*-means achieves our objective of clustering big data sets efficiently and accurately.

Chapter 3

Kernel-based Clustering Using Random Feature Maps

3.1 Introduction

Although the approximate kernel *k*-means algorithm is accurate and scalable, it has the following limitations:

- The approximate kernel *k*-means algorithm samples a subset of m points from the data set, and constructs a $n \times m$ kernel matrix K_B, between the n points in the data set and the sampled points. When n is in the order of billions, and the number of clusters is also comparably large, calculating the $O(nm)$ matrix K_B may be infeasible. For instance, if we were to cluster the Tiny image data set containing 80 million images into $75,062$ clusters (the true number of classes in the data set), approximate kernel *k*-means would require about $m = 10^5$ samples. This would boil down to calculating about 8 trillion similarity values, which is computationally expensive.

- Approximate kernel *k*-means cannot efficiently handle out-of-sample clustering, i.e. the problem of assigning new data points to clusters after the clustering is complete. In order to

find the cluster label for a new point $\mathbf{x}^*$, we need to compute

$$||\mathbf{c}_k(\cdot) - \kappa(\mathbf{x}^*, \cdot)||^2_{\mathcal{H}_\kappa} = \alpha_k^\top \widehat{K} \alpha_k - 2\varphi^\top \alpha_k, \; k \in [C],$$

where $\varphi = [\kappa(\mathbf{x}^*, \widehat{\mathbf{x}}_1), \ldots, \kappa(\mathbf{x}^*, \widehat{\mathbf{x}}_m)]$ and α_k is the k^{th} row of the $C \times m$ matrix α, containing the weights of the sampled points in each of the C clusters. This operation has $O(m^2C + mC^2 + md)$ running time complexity and can be inefficient for large m.

To address the above limitations, we propose two algorithms which use random feature maps to obtain an $O(m)$-dimensional embedding of the Hilbert space associated with the kernel $\kappa(\cdot, \cdot)$, where $m \ll n$ [42]. Our first algorithm called the *RFF clustering* algorithm obtains vector representations of the data points to form an $n \times m$ pattern matrix. This pattern matrix is clustered using a linear clustering algorithm like *k*-means to obtain the data partitions. This algorithm, like the approximate kernel k-means, has $O(nm)$ running time complexity and memory requirements. The second algorithm, which we call the *SV clustering* algorithm, is designed along the lines of spectral clustering. It approximates the eigenvectors of the $n \times n$ kernel matrix by the dominant C singular vectors of the pattern matrix, and obtains the data partition by clustering these singular vectors in $O(nC^2)$ time. The SV clustering algorithm provides a C-dimensional representation of the cluster centers, using which previously unseen data points can be assigned to clusters efficiently.

3.2 Background

The matrix approximation methods discussed in Section 2.2 essentially factorize the kernel matrix to obtain a low-dimensional representation of the data. Another form of kernel approximation, initially proposed for supervised kernel-based learning by Rahimi and Recht in [147], involves factorizing the kernel function instead of the kernel matrix, by mapping the data explicitly into a low-dimensional randomized feature space.

A kernel function $\kappa(\cdot, \cdot)$ is *shift-invariant* if $\kappa(\mathbf{x}, \mathbf{y}) = \kappa(\mathbf{x} - \mathbf{y})$ for all $\mathbf{x}, \mathbf{y} \in \Re^d$. Popular examples of shift-invariant kernels are the RBF and Laplacian kernels. Let $p(\mathbf{w})$ denote the Fourier transform of such a kernel function $\kappa(\mathbf{x} - \mathbf{y})$, i.e.

$$\kappa(\mathbf{x} - \mathbf{y}) = \int_{\Re^d} p(\mathbf{w}) \exp(j\mathbf{w}^\top(\mathbf{x} - \mathbf{y}))d\mathbf{w}.$$

According to the following theorem from harmonic analysis, $p(\mathbf{w})$ is a valid probability density function, provided the kernel function is continuous, positive-definite and scaled appropriately.

Theorem 2. *(Bochner's theorem [152]) A continuous kernel $\kappa(\mathbf{x}, \mathbf{y}) = \kappa(\mathbf{x} - \mathbf{y})$ on $\Re^d$ is positive definite if and only if $\kappa(\delta)$ is the Fourier transform of a non-negative measure.*

For instance, the Fourier transform [26] of the RBF kernel function is the Gaussian probability distribution function. Let $\mathbf{w}$ be a d-dimensional vector sampled from $p(\mathbf{w})$. The kernel function can be approximated as

$$\kappa(\mathbf{x}, \mathbf{y}) = \mathbb{E}_{\mathbf{w}}\left[f(\mathbf{w}, \mathbf{x})^\top f(\mathbf{w}, \mathbf{y})\right], \tag{3.1}$$

where

$$f(\mathbf{w}, \mathbf{x}) = (\cos(\mathbf{w}^\top\mathbf{x}), \sin(\mathbf{w}^\top\mathbf{x}))^\top.$$

We can approximate the expectation in (3.1) with the empirical mean over m Fourier components $\{\mathbf{w}_1, \ldots, \mathbf{w}_m\}$, sampled from the distribution $p(\mathbf{w})$, and obtain the following representation for the point $\mathbf{x}$:

$$\mathbf{z}(\mathbf{x}) = \frac{1}{\sqrt{m}}(\cos(\mathbf{w}_1^\top\mathbf{x}), \ldots, \cos(\mathbf{w}_m^\top\mathbf{x}), \sin(\mathbf{w}_1^\top\mathbf{x}), \ldots, \sin(\mathbf{w}_m^\top\mathbf{x})). \tag{3.2}$$

The features $\mathbf{z}(\mathbf{x})$ are called the *Random Fourier Features*. The kernel similarity between any two points $\mathbf{x}$ and $\mathbf{y}$ can be approximated by the inner product between the random Fourier features

Algorithm 6 RFF Clustering

1: **Input**:
   - $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}, \mathbf{x}_i \in \Re^d$: the set of n d-dimensional data points to be clustered clustered
   - λ: the RBF kernel width parameter
   - C: the number of clusters
   - m: the number of Fourier components ($C < m \ll n$)
2: **Output**: Cluster membership matrix $U \in \{0,1\}^{C \times n}$
3: Draw m independent samples $\mathbf{w}_1, \ldots, \mathbf{w}_m$ from the Gaussian distribution $\mathcal{N}\left(0, \frac{1}{\lambda} I\right)$. Let $W = (\mathbf{w}_1, \ldots, \mathbf{w}_m)$.
4: Compute the matrix $H = [\cos(XW) \sin(XW)]$, where $X = (\mathbf{x}_1, \ldots, \mathbf{x}_n)^\top$ is the input pattern matrix.
5: Run the *k*-means algorithm (Algorithm 1) on H with the number of clusters set to C, and obtain the membership matrix U.

corresponding to the data points, i.e.

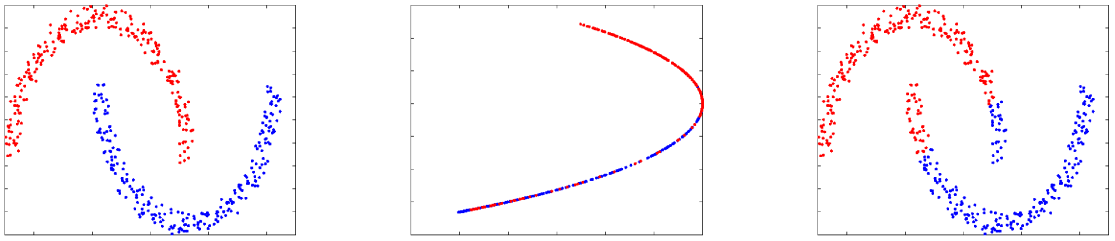
$$\kappa(\mathbf{x}, \mathbf{y}) \simeq \mathbf{z}(\mathbf{x})^\top \mathbf{z}(\mathbf{y}). \tag{3.3}$$

Given a data set $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, we can obtain its low-dimensional representation $\widehat{\mathcal{D}} = \{\mathbf{z}(\mathbf{x}_1), \ldots, \mathbf{z}(\mathbf{x}_n)\}$, and apply a fast linear learning algorithm to $\widehat{\mathcal{D}}$, instead of executing a kernel-based learning algorithm on $\mathcal{D}$. This allows us to learn the non-linear relations in the data efficiently using linear machines.

This kernel approximation has been employed in several large-scale learning tasks such as classification [25, 147, 182], regression [123], data compression [146] and novelty detection [164]. Random feature maps have been extended to shift-variant kernels such as intersection kernels [110, 179] and other positive definite kernels using Maclaurin and Taylor expansions of the kernel function [81, 92].

3.3 Kernel Clustering using Random Fourier Features

Random feature maps can be used for clustering big data sets efficiently. We propose an algo-



(a) (b) (c)

Figure 3.1 A simple example to illustrate the RFF clustering algorithm. (a) Two-dimensional data set with 500 points from two clusters (250 points in each cluster), (b) Plot of the matrix H obtained by sampling $m = 1$ Fourier component. (c) Clusters obtained by executing *k*-means on H.

Table 3.1 Comparison of the confusion matrices of the RFF, kernel *k*-means, and *k*-means algorithms for the two-dimensional semi-circles data set, containing 500 points (250 points in each of the two clusters).

	Class 1	**Class 2**
Cluster 1	220	41
Cluster 2	30	209

(a) RFF clustering

	Class 1	**Class 2**
Cluster 1	250	0
Cluster 2	0	250

(b) Kernel *k*-means

	Class 1	**Class 2**
Cluster 1	132	129
Cluster 2	118	121

(c) *k*-means

rithm called the *RFF clustering* algorithm, which first projects the data set into a low-dimensional space using random Fourier feature maps, and then executes *k*-means on the transformed data.

Let $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ represent the input data set, and $\kappa(\cdot, \cdot)$ be the kernel function. We assume that $\kappa(\cdot, \cdot)$ is shift-invariant[1] and satisfies the condition $\kappa(\mathbf{x}, \mathbf{x}) = \kappa(0) = 1$. Let $K = [\kappa(\mathbf{x}_i, \mathbf{x}_j)]_{n \times n}$ denote the kernel matrix. The matrix

$$H = \left[\mathbf{z}(\mathbf{x}_1)^\top, \ldots, \mathbf{z}(\mathbf{x}_n)^\top\right] \tag{3.4}$$

denotes the data matrix obtained by mapping each point $\mathbf{x} \in \mathcal{D}$ using the random feature map $\mathbf{z}(\cdot)$.

[1]The assumption of shift-invariance is made only for simplicity. Random feature maps can be used for other positive semi-definite kernels as well, as demonstrated in [81, 92].

Using (3.3), we can approximate the kernel matrix K by

$$\widehat{K} = H^\top H. \tag{3.5}$$

We can replace the kernel matrix K in the kernel *k*-means optimization problem (1.11) with the approximate kernel matrix $\widehat{K}$ in (3.5), leading to the following optimization problem:

$$\max_{U \in \mathcal{P}} \text{tr}(\widetilde{U} H^\top H \widetilde{U}^\top), \tag{3.6}$$

where $U = (\mathbf{u}_1, \ldots, \mathbf{u}_C)^\top$ is the cluster membership matrix, $\mathcal{P} = \{U \in \{0,1\}^{C \times n} : U^\top \mathbf{1} = \mathbf{1}\}$, $\widetilde{U} = [\text{diag}\,(U\mathbf{1})]^{-1/2}\, U$, and $\mathbf{1}$ is a vector of all ones. By comparing the above problem to the *k*-means optimization problem (1.12), it becomes evident that the problem in (3.6) can be solved by executing *k*-means on the matrix H. Algorithm 6 describes the RFF clustering algorithm for clustering using the random Fourier features obtained from the RBF kernel. We illustrate the algorithm in Figure 3.1. Figure 3.1(a) shows a two-dimensional data set containing 500 points from two semi-circular clusters. The two clusters are identified perfectly when the kernel *k*-means algorithm is executed on this data set. For the purpose of illustration, we sampled one Fourier component (i.e. $m = 1$) and generated a two-dimensional matrix H to represent the data. A plot of this representation is shown in Figure 3.1(b). Note that the two clusters are more separated in this space than in the original feature space. Figure 3.1(c) shows the clusters obtained when *k*-means is executed on H. The error, in terms of the number of points that are grouped into the wrong cluster, is about 14%, as shown in the confusion matrices in Table 3.1. A confusion matrix shows the mapping between the true class labels and the cluster labels. Each cluster is assigned a class label, corresponding to the true label of the majority of the data points in the cluster. Each entry (k, c) in the confusion matrix represent the number of data points from class c assigned to cluster k. The diagonal entries represent the number of points that have been assigned to the correct cluster. The confusion matrices show that the accuracy of the RFF clustering algorithm is close to that of the

kernel *k*-means algorithm, and higher than that of the *k*-means algorithm.

3.3.1 Analysis

In this section, we first analyze the computational complexity of the RFF clustering algorithm, and then examine the quality of the data partitions generated.

3.3.1.1 Computational complexity

Sampling from the Fourier transform of the kernel function is a relatively inexpensive operation for most shift-invariant kernels. For instance, several efficient techniques have been proposed for sampling from a Gaussian distribution in the literature [53]. The crux of the proposed RFF clustering algorithm thus lies in computing the low-dimensional random Fourier features H. Given m d-dimensional Fourier components, the mapping to the matrix H can be performed in $O(ndm)$ time. Le *et al.* proposed the Fastfood algorithm which reduces the running time complexity of this operation to $O(nm\log(d))$ [107]. Instead of directly multiplying the data matrix X with the random Gaussian matrix W to obtain the matrix H, they combine W with a Walsh-Hadamard matrix. Multiplication with Hadamard matrices can be performed in loglinear time, thereby reducing the running time. As Gaussian matrices combined with Hadamard matrices behave like Gaussian matrices, this does not affect the kernel matrix approximation significantly. Executing *k*-means on H takes $O(nmCl)$ time, where l is the number of iterations required for convergence. Thus, the overall running time complexity of the RFF clustering algorithm is $O(nm\log(d) + nmCl)$. Only $O(nm)$ memory is required to store the matrix H.

3.3.1.2 Approximate error

To examine the difference between the clustering solutions of the kernel *k*-means algorithm and the RFF clustering algorithm, we must first bound the kernel approximation error $\left|\left|K - \widehat{K}\right|\right|_F$. In the following theorem, we show that this error decreases at the rate of $O(1/\sqrt{m})$:

Theorem 3. *For any $\delta \in (0,1)$, with probability $1-\delta$, we have*

$$\left|\left|\widehat{K} - K\right|\right|_F \leq \frac{2\ln(2/\delta)}{m} + \sqrt{\frac{2\ln(2/\delta)}{m}} = O\left(\frac{1}{\sqrt{m}}\right). \tag{3.7}$$

Proof. We use the following result from [165] to prove this theorem:

Lemma 4. *Let $\mathcal{H}_\kappa$ be a Hilbert space and ξ be a random variable on (Z,ρ) with values in $\mathcal{H}_\kappa$. Assume $\|\xi\| \leq M < \infty$ almost surely. Denote $\sigma^2(\xi) = \mathbb{E}(\|\xi\|^2)$. Let $\{z_i\}_{i=1}^m$ be independent random drawers of ρ. For any $0 < \delta < 1$, with confidence $1-\delta$,*

$$\left\|\frac{1}{m}\sum_{i=1}^{m}(\xi_i - \mathbb{E}[\xi_i])\right\| \leq \frac{2M\ln(2/\delta)}{m} + \sqrt{\frac{2\sigma^2(\xi)\ln(2/\delta)}{m}}. \tag{3.8}$$

Define

$$\begin{aligned}
\mathbf{a}(\mathbf{w}) &= \frac{1}{\sqrt{n}}(\cos(\mathbf{w}^\top\mathbf{x}_1),\ldots,\cos(\mathbf{w}^\top\mathbf{x}_n))^\top \text{ and} \\
\mathbf{b}(\mathbf{w}) &= \frac{1}{\sqrt{n}}(\sin(\mathbf{w}^\top\mathbf{x}_1),\ldots,\sin(\mathbf{w}^\top\mathbf{x}_n))^\top.
\end{aligned}$$

Let $\xi_i = \mathbf{a}(\mathbf{w}_i)\mathbf{a}(\mathbf{w}_i)^\top + \mathbf{b}(\mathbf{w}_i)\mathbf{b}(\mathbf{w}_i)^\top$. We have $\mathbb{E}[\xi_i] = \mathbb{E}[\mathbf{a}(\mathbf{w}_i)\mathbf{a}(\mathbf{w}_i)^\top + \mathbf{b}(\mathbf{w}_i)\mathbf{b}(\mathbf{w}_i)^\top] = K$ and $||\xi_i||_F^2 = ||\mathbf{a}(\mathbf{w}_i)|^2 + |\mathbf{b}(\mathbf{w}_i)||^2 = 1$, which implies $M = \sigma^2 = 1$. We obtain the result (3.7) by substituting these values in (3.8). □

$\widehat{K}$ is a good approximation of K provided that the number of Fourier components m is sufficiently large. We can now obtain an upper bound on the difference between the solutions of the kernel *k*-means optimization problem in (1.11) and the optimization problem in (3.6):

Theorem 4. *Let U^* and U_m^* be the optimal solutions of* (1.11) *and* (3.6)*, respectively. Let $\widetilde{U}^* = U^*[D^*]^{-1/2}$ and $\widetilde{U}_m^* = U_m^*[D_m^*]^{-1/2}$ denote the normalized versions of U^* and U_m^*, where $D^* =$*

diag$([U^*]^\top \mathbf{1})$ *and* $D_m^* = $ *diag*$([U_m^*]^\top \mathbf{1})$. *For any* $\delta \in (0,1)$*, with probability* $1-\delta$*, we have*

$$\begin{aligned} \textit{tr}\left([\widetilde{U}^* - \widetilde{U}_m^*]^\top K [\widetilde{U}^* - \widetilde{U}_m^*]\right) &\leq \frac{4\ln(2/\delta)}{m} + \sqrt{\frac{8\ln(2/\delta)}{m}} \\ &= O\left(\frac{1}{\sqrt{m}}\right). \end{aligned}$$

Proof. We have

$$\begin{aligned} \text{tr}([\widetilde{U}^*]^\top K \widetilde{U}^*) &\leq \text{tr}([\widetilde{U}^*]^\top \widehat{K} \widetilde{U}^*) + \left|\left|K - \widehat{K}\right|\right|_F \\ &\leq \text{tr}([\widetilde{U}_m^*]^\top \widehat{K} \widetilde{U}_m^*) + \left|\left|K - \widehat{K}\right|\right|_F \\ &\leq \text{tr}([\widetilde{U}_m^*]^\top K \widetilde{U}_m^*) + 2\left|\left|K - \widehat{K}\right|\right|_F. \end{aligned}$$

Since $\text{tr}([\widetilde{U}^*]^\top K \widetilde{U}^*) \geq \text{tr}([\widetilde{U}_m^*]^\top K \widetilde{U}_m^*)$, we have

$$|\text{tr}([\widetilde{U}^*]^\top K \widetilde{U}^*) - \text{tr}([\widetilde{U}_m^*]^\top K \widetilde{U}_m^*)| \leq 2\left|\left|K - \widehat{K}\right|\right|_F.$$

We complete the proof by using the result from Theorem 3 and the strong convexity property of $\text{tr}(\widetilde{U}^\top K \widetilde{U})$. □

3.4 Kernel Clustering using Random Fourier Features in Constrained Eigenspace

Despite its simplicity, RFF clustering may suffer from high computational cost. As seen in Theorem 4, a large number of random Fourier components may be required to achieve a low approximation error. As a consequence, we need to execute *k*-means over a high-dimensional space, leading to high runtime complexity. To address this problem, we propose using an idea similar to that in the approximate kernel *k*-means algorithm, and constrain the cluster centers to lie in the

Algorithm 7 SV Clustering

1: **Input**:
   - $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}, \mathbf{x}_i \in \Re^d$: the set of n d-dimensional data points to be clustered
   - λ: the RBF kernel width parameter
   - C: the number of clusters
   - m: the number of Fourier components ($C < m \ll n$)
2: **Output**: Cluster membership matrix $U \in \{0,1\}^{C \times n}$
3: Draw m independent samples $\mathbf{w}_1, \ldots, \mathbf{w}_m$ from the Gaussian distribution $\mathcal{N}\left(0, \frac{1}{\lambda} I\right)$. Let $W = (\mathbf{w}_1, \ldots, \mathbf{w}_m)$.
4: Compute the matrix $H = [\cos(XW) \sin(XW)]$, where $X = (\mathbf{x}_1, \ldots, \mathbf{x}_n)^\top$ is the input pattern matrix.
5: Compute the left singular vectors of H corresponding to its top C singular values to obtain the matrix $\widehat{V}_C = (\widehat{\mathbf{v}}_1, \ldots, \widehat{\mathbf{v}}_C)$.
6: Run the *k*-means algorithm (Algorithm 1) on $\widehat{V}_C$ with the number of clusters set to C and obtain the membership matrix U.

subspace spanned by the top eigenvectors of the kernel matrix. Let $\{(\lambda_i, \mathbf{v}_i)\}_{i=1}^n$ denote the eigenvalues and eigenvectors of the kernel matrix K, ranked in the descending order of the eigenvalues. Let $\mathcal{H}_a = \text{span}(\mathbf{v}_1, \ldots, \mathbf{v}_C)$ represent the space spanned by the dominant C eigenvectors. The kernel *k*-means problem in (1.7) can be approximated as

$$\min_{U \in \mathcal{P}} \max_{\{\mathbf{c}_k(\cdot) \in \mathcal{H}_a\}_{k=1}^C} \sum_{k=1}^{C} \sum_{i=1}^{n} \frac{U_{ki}}{n} ||\mathbf{c}_k(\cdot) - \kappa(\mathbf{x}_i, \cdot)||^2_{\mathcal{H}_\kappa}, \tag{3.9}$$

where $\mathbf{c}_k(\cdot)$ represent the cluster centers, $U = (\mathbf{u}_1, \ldots, \mathbf{u}_C)^\top$ is the cluster membership matrix, $\mathcal{P} = \{U \in \{0,1\}^{C \times n} : U^\top \mathbf{1} = \mathbf{1}\}$, and $\mathbf{1}$ is a vector of all ones. The above problem (3.9) can be solved by executing *k*-means on the top eigenvectors of K, i.e. by solving the following optimization problem:

$$\max_{U \in \mathcal{P}} \text{tr}(\widetilde{U}[V_C V_C^\top]\widetilde{U}^\top), \tag{3.10}$$

where $V_C = (\mathbf{v}_1, \ldots, \mathbf{v}_C)$, and $\widetilde{U} = [\text{diag}(U\mathbf{1})]^{-1/2} U$. This method leads to a significant reduction in computational cost when compared to the RFF clustering algorithm, as each data point is represented by a C-dimensional vector and *k*-means needs to be executed over a lower dimensional space.

However, computing the eigenvectors of K requires the computation of the $n \times n$ kernel matrix, which is infeasible when n is large. We circumvent this issue by approximating the eigenvectors of K using the singular vectors of the random Fourier features, and thereby avoid computing the full kernel matrix. More specifically, we compute the top C singular values and the corresponding left singular vectors of H, denoted by $\{(\widehat{\lambda}_i, \widehat{\mathbf{v}}_i)\}_{i=1}^{C}$, and represent the data points in $\mathcal{D}$ by the matrix $\widehat{V}_C = (\widehat{\mathbf{v}}_1, \ldots, \widehat{\mathbf{v}}_C)$. We then solve the approximate optimization problem

$$\max_{U \in \mathcal{P}} \mathbf{tr}(\widetilde{U}[\widehat{V}_C \widehat{V}_C^\top]\widetilde{U}^\top), \tag{3.11}$$

by executing *k*-means on the matrix $\widehat{V}_C$ to obtain the C clusters. This procedure, named as the SV clustering algorithm, is outlined in Algorithm 7. It has the same input and output as the RFF clustering algorithm, but differs in the final two steps. As the dimensionality of the input to the *k*-means clustering step in the SV clustering algorithm is significantly smaller than that in the RFF clustering algorithm, SV clustering is more efficient than RFF clustering, despite the overhead of computing the singular vectors.

3.4.1 Analysis

In this section, we discuss the computational complexity of the SV clustering algorithm and bound its approximation error.

3.4.1.1 Computational complexity

As the initial steps in the SV clustering algorithm are the same as the RFF clustering algorithm, these steps have the same running time complexity. In addition, the algorithm involves performing the singular value decomposition of H. If the top singular vectors of H are found using conventional methods, the runtime complexity of the SVD step would be $O(nm^2)$. We reduce this complexity in our implementation by using the approximate SVD technique proposed in [70]. We

sample s rows from H to form an $s \times 2m$ matrix S. The top eigenvectors of $S^\top S$, denoted by $\widetilde{V} = (\widetilde{\mathbf{v}}_1, \ldots, \widetilde{\mathbf{v}}_C)$, are close to the top eigenvectors of $H^\top H$ and the singular vectors of H can be recovered from the eigenvectors of $S^\top S$ as $H\widetilde{V}$. Using this approximation, the runtime complexity of SVD is reduced to $O\left(sm \min\{s, m\}\right)$. The time taken to execute *k*-means on the singular vectors is $O(nC^2l)$.

When $\max(m, s, l, C) \ll n$, both the RFF and SV clustering algorithms have linear time complexity. However, the time taken by the *k*-means step in the SV clustering algorithm is $O(nC^2l)$, as opposed to $O(nmCl)$, the time taken by the *k*-means step in the RFF clustering algorithm. As C is usually much lesser than m, the SV algorithm is much more efficient than the RFF clustering algorithm.

The values chosen for m and s introduce a trade-off between the clustering quality and efficiency. Higher values result in better clustering quality but lesser speedup. In our implementation, we found that a reasonably good accuracy can be achieved by setting the value of m to range between 1% and 2% of n, and setting s to around 2% of n. Lower m/n ratio values work well as n increases.

3.4.1.2 Approximation error

The SV clustering algorithm relies on the assumption of the existence of a large eigengap. This theory that has been adopted by many earlier kernel-based algorithms which rely on the spectral embedding of the data [118], essentially implies that most attributes of the data can be well approximated by vectors in the low-dimensional space spanned by the top eigenvectors.

The following theorem proves that when the last $n - C$ eigenvalues $\{\lambda_i\}_{i=C+1}^{n}$ of K are sufficiently small, the subspace $\mathcal{H}$ can be well approximated by the subspace $\mathcal{H}_a$ spanned by the top C eigenvectors of K.

Theorem 5. *Let E and E_a represent the optimal clustering errors in the kernel k-means problem*

(1.7) *and the optimization problem* (3.9)*, respectively. We have*

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

Proof. Let $\{\mathbf{c}_k^*(\cdot)\}_{k=1}^C$ and U^* be the optimal solutions to (1.7). Let $c_k^a(\cdot)$ represent the projection of c_k^* into the subspace $\mathcal{H}_a$. For any $\kappa(\mathbf{x}_i, \cdot)$, let $g_i(\cdot)$ and $h_i(\cdot)$ be the projections of $\kappa(\mathbf{x}_i, \cdot)$ into the subspace $\mathcal{H}_a$ and span$(\mathbf{v}_{C+1}, \ldots, \mathbf{v}_n)$, respectively. We have

$$\begin{aligned}
E_a &= \min_U \max_{\mathbf{c}_k(\cdot) \in \mathcal{H}_a} \sum_{k=1}^{C} \sum_{i=1}^{n} \frac{U_{ki}}{n} ||\mathbf{c}_k(\cdot) - \kappa(\mathbf{x}_i, \cdot)||_{\mathcal{H}_\kappa}^2 \\
&\leq \sum_{k=1}^{C} \sum_{i=1}^{n} \frac{U_{ki}^*}{n} ||\mathbf{c}_k^a(\cdot) - \kappa(\mathbf{x}_i, \cdot)||_{\mathcal{H}_\kappa}^2 \\
&= \sum_{k=1}^{C} \sum_{i=1}^{n} \frac{U_{ki}^*}{n} \left(||c_k^a(\cdot) - g_i(\cdot)||_{\mathcal{H}_\kappa}^2 + ||h_i(\cdot)||_{\mathcal{H}_\kappa}^2 \right) \\
&\leq E + \frac{1}{n} \sum_{k=1}^{C} \sum_{i=1}^{n} ||h_i(\cdot)||_{\mathcal{H}_\kappa}^2 \\
&\leq E + \sum_{i=C+1}^{n} \lambda_i.
\end{aligned}$$

□

We prove a set of preliminary lemmas before presenting our main result in Theorem 6 which bounds the clustering error of the SV clustering algorithm.

Lemma 5. *(Result from matrix perturbation theory [166]) Let $(\lambda_i, \mathbf{v}_i), i \in [n]$ be the eigenvalues and eigenvectors of a symmetric matrix $A \in \Re^{n \times n}$ ranked in the descending order of eigenvalues. Set $X = (\mathbf{v}_1, \ldots, \mathbf{v}_C)$ and $Y = (\mathbf{v}_{C+1}, \ldots, \mathbf{v}_n)$. Given a symmetric perturbation matrix E, let*

$$(X, Y)^\top E (X, Y) = \begin{pmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \end{pmatrix}.$$

Let $||\cdot||$ represent a consistent family of norms and let

$$\gamma = ||E_{21}||\,, \delta = \lambda_C - \lambda_{C+1} - ||E_{11}|| - ||E_{22}||\,.$$

If $\delta > 0$ and $\frac{\gamma}{\delta} < \frac{1}{2}$, then there exists a unique matrix $P \in \Re^{(n-C)\times C}$ satisfying

$$||P|| < \frac{2\gamma}{\delta},$$

such that

$$\begin{aligned} X' &= (X + YP)(I + P^\top P)^{-1/2}, \text{ and} \\ Y' &= (Y - XP^\top)(I + PP^\top)^{-1/2} \end{aligned}$$

are the eigenvectors of $A + E$.

Lemma 6. *Given $\delta \in (0, 1)$, we assume $(\lambda_C - \lambda_{C+1}) \geq 3\Delta$, where*

$$\Delta = \frac{2\ln(2/\delta)}{m} + \sqrt{\frac{2\ln(2/\delta)}{m}}, \tag{3.12}$$

there exists, with probability $1 - \delta$, a matrix $P \in \Re^{(n-C)\times C}$ satisfying

$$||P||_F \leq \frac{2\Delta}{\lambda_C - \lambda_{C+1} - \Delta},$$

such that

$$\widehat{V}_C = (V_C + \bar{V}_C P)(I + P^\top P)^{-1/2},$$

where $\widehat{V}_C = (\widehat{\mathbf{v}}_1, \ldots, \widehat{\mathbf{v}}_C)$, $V_C = (\mathbf{v}_1, \ldots, \mathbf{v}_C)$, and $\bar{V}_C = (\mathbf{v}_{C+1}, \ldots, \mathbf{v}_n)$.

Proof. Let $E = \widehat{K} - K$. Using Theorem 3 and Lemma 5, we have

$$\gamma = \left|\left|V_C^\top E \bar{V}_C\right|\right| \leq ||E|| \leq \Delta,$$

and

$$\begin{aligned} \delta &= \lambda_C - \lambda_{C+1} - \left|\left|V_C^\top E V_C\right|\right| - \left|\left|\bar{V}_C^\top E \bar{V}_C\right|\right| \\ &\geq \lambda_C - \lambda_{C+1} - \Delta > 0. \end{aligned}$$

As $\lambda_C - \lambda_{C+1} \geq 3\Delta$, we also have $\frac{\gamma}{\delta} < \frac{1}{2}$, allowing us to apply Lemma 5 and obtain the required result. □

Lemma 7. *Under the assumptions of Lemma 6, with a probability* $1 - \delta$*, we have*

$$\sum_{i=1}^{C} ||\widehat{\mathbf{v}}_i - \mathbf{v}_i||^2 \leq 2\,||P||_F^2 \leq \frac{18\Delta^2}{(\lambda_C - \lambda_{C+1})^2},$$

where Δ *is defined in* (3.12)*.*

Proof. Define $A = P(I + P^\top P)^{-1/2}$ and $B = I - (I + P^\top P)^{-1/2}$. Let $\{\gamma_i\}_{i=1}^C$ be the eigenvalues of $P^\top P$. Using the result from Lemma 6, we have

$$\begin{aligned} \sum_{i=1}^{C} ||\widehat{\mathbf{v}}_i - \mathbf{v}_i||^2 &= \left|\left|\bar{V}_C A\right|\right|_F^2 + ||V_C B||_F^2 \\ &\leq ||A||_F^2 + ||B||_F^2 \\ &\leq ||P||_F^2 + \sum_{i=1}^{C} \frac{\gamma_i}{(1 + \sqrt{\gamma_i})^2} \\ &\leq 2\,||P||_F^2\,. \end{aligned}$$

We complete the proof by using the fact that

$$||P||_F \leq \frac{2\Delta}{\lambda_C - \lambda_{C+1} - \Delta} \leq \frac{3\Delta}{\lambda_C - \lambda_{C+1}}.$$

□

In the following theorem, we bound the approximation error of the SV clustering algorithm and show that it yields a better approximation of kernel clustering than the RFF clustering algorithm, provided there is a sufficiently large gap in the eigenspectrum.

Theorem 6. *Let U^* and U_m^* be the optimal solutions of* (3.10) *and* (3.11)*, and let $\widetilde{U}^*$ and $\widetilde{U}_m^*$ represent their normalized versions (as defined in Theorem 4), respectively. Given $\delta \in (0, 1)$, assume $(\lambda_C - \lambda_{C+1}) \geq 3\Delta$, where Δ is defined in* (3.12)*. With probability $1 - \delta$, we have*

$$tr\left([\widetilde{U}^* - \widetilde{U}_m^*]^\top [\widetilde{U}^* - \widetilde{U}_m^*]\right) \leq \frac{18\Delta^2}{(\lambda_C - \lambda_{C+1})^2} = O\left(\frac{1}{m}\right).$$

Proof. This theorem is a direct result of Lemmas 6 and 7. □

Theorem 6 shows that, like the RFF clustering algorithm, the SV clustering algorithm's approximation error reduces as the number of Fourier components increases, albeit at a higher rate of $O(1/m)$.

3.4.2 Out-of-sample Clustering

The SV clustering algorithm can be used to efficiently assign cluster labels to data points that were not seen previously. The cluster centers in the SV clustering algorithm lie in the subspace $\mathcal{H}_a = \text{span}(\widehat{\mathbf{v}}_1, \ldots, \widehat{\mathbf{v}}_C)$, and can be expressed as linear combinations of these vectors:

$$\widetilde{\mathbf{c}}_k = \frac{1}{n_k} \sum_{i=1}^{n} U_{ki} \widehat{\mathbf{v}}_i,$$

where n_k is the number of data points in the k^{th} cluster. Given a data point $\mathbf{x}^* \in \Re^d$, we can obtain its cluster label using the following double projection scheme:

(i) Compute the random Fourier features

$$z(\mathbf{x}^*) = \frac{1}{\sqrt{m}}(\cos(\mathbf{w}_1^\top \mathbf{x}^*), \ldots, \cos(\mathbf{w}_m^\top \mathbf{x}^*), \sin(\mathbf{w}_1^\top \mathbf{x}^*), \ldots, \sin(\mathbf{w}_m^\top \mathbf{x}^*)).$$

(ii) Project $z(\mathbf{x}^*)$ into the subspace $\mathcal{H}_a$ to obtain $\widehat{\mathbf{v}}^*$.

(iii) Assign $\mathbf{x}^*$ to the cluster k which minimizes $||\widetilde{\mathbf{c}}_k - \widehat{\mathbf{v}}^*||_2^2$.

Using this process, cluster labels can be assigned in $O(md)$ time.

3.5 Experimental Results

3.5.1 Data sets

We evaluated the performance of the RFF and SV clustering algorithms on the CIFAR-10, MNIST, Forest Cover Type, Imagenet-34, Poker, and Network Intrusion data sets. The medium-sized CIFAR-10 and MNIST data sets are used to compare the performance of the proposed algorithms with the kernel *k*-means algorithm. The remaining data sets are used to demonstrate the scalability of the algorithms to large data sets.

3.5.2 Baselines

Using the medium-sized CIFAR-10 and MNIST data sets, we compared the proposed algorithms with the kernel *k*-means algorithm, to demonstrate that their clustering performance is close to that of the kernel *k*-means in terms of cluster quality. We also compared their performance with the approximate kernel *k*-means algorithm and the Nystrom approximation based spectral clustering

algorithm. We also gauged the performance of our algorithms against that of the *k*-means algorithm to show that they achieve better cluster quality.

3.5.3 Parameters

We used the RBF kernel for all the kernel-based algorithms on all the data sets. We set the kernel width equal to $\rho\mathfrak{d}$, where $\mathfrak{d}$ is the average pairwise Euclidean distance between the data points and parameter ρ is tuned in the range $[0, 1]$ to obtain optimal performance[2]. We varied the number of Fourier components m from 100 to $2,000$. For the approximate kernel *k*-means and spectral clustering algorithms, m represents the size of the sample drawn from the data set. The value of s, the number of rows sampled from H to compute the approximate singular vectors, was set to 2% of the total number of data points n. The number of clusters C was set equal to the true number of classes in the data set.

All algorithms were implemented in MATLAB[3] and run on a 2.8 GHz processor using 40 GB RAM. All results are averaged over 10 runs of the algorithms. In each run of the proposed algorithms, we used a different set of randomly sampled Fourier components. For the baseline algorithms which use a subset of the data, we used different randomly sampled subsets in each run.

3.5.4 Results

3.5.4.1 Running time

The running time of the baseline algorithms and the proposed RFF and SV algorithms are recorded in Table 3.2. The number of Fourier components m for the RFF and SV clustering algorithms

[2]The average pairwise similarity was used only as a heuristic to set the RBF kernel width, and not required by the proposed algorithms. Other techniques may be employed to choose the kernel and the kernel parameters.

[3]We used the *k*-means implementation in the MATLAB Statistics Toolbox and the Nystrom approximation based spectral clustering implementation [35] available at http://alumni.cs.ucsb.edu/ wychen/sc.html. The remaining algorithms were implemented in-house.

Table 3.2 Running time (in seconds) of the RFF and SV clustering algorithms on the six benchmark data sets. The parameter m, which represents the number of Fourier components for the RFF and SV clustering algorithms, and the sample size for the approximate kernel *k*-means and Nystrom approximation based spectral clustering algorithms, is set to $m = 2,000$. It is not feasible to execute kernel *k*-means on the large Forest Cover Type, Imagenet-34, Poker, and Network Intrusion data sets due to their large size. An approximate of the running time of kernel *k*-means on these data sets is obtained by first executing kernel *k*-means on a randomly chosen subset of $50,000$ data points to find the cluster centers, and then assigning the remaining points to the closest cluster center.

Data set	**RFF clustering (proposed)**	**SV clustering (proposed)**	**Approx. kernel *k*-means**	**Nystrom approx. based spectral clustering**	**Kernel *k*-means**	***k*-means**
CIFAR-10	3,418.21 (± 907.14)	58.32 (± 38.68)	37.01 (± 6.52)	116.13 (± 1.97)	725.32 (± 7.39)	159.22 (± 75.81)
MNIST	1,089.26 (± 483.63)	39.94 (± 5.64)	57.73 (± 12.94)	4,186.02 (± 386.17)	914.59 (± 235.14)	448.69 (± 177.24)
Forest Cover Type	2,078.63 (± 617.22)	76.99 (± 17.04)	157.48 (± 27.37)	573.55 (± 327.49)	4,721.03 (± 504.21)	40.88 (± 6.4)
Imagenet-34	1,333.85 (± 6.53)	212.32 (± 4.75)	1,261.02 (± 37.39)	1,841.47 (± 123.82)	154,416 ($\pm 32,302$)	31,076 ($\pm 9,355$)
Poker	4,530.44 (± 276.37)	41.08 (± 2.57)	256.26 (± 44.84)	520.48 (± 51.29)	9,942 ($\pm 1,476$)	40.88 (± 6.40)
Network Intrusion	24,151 ($\pm 6,351.34$)	435.53 (± 189.07)	891.08 (± 237.17)	1,682.46 (± 235.70)	34,784 ($\pm 1,493$)	953.41 (± 169.38)

was set to $2,000$ and the sample set size for the approximate kernel *k*-means and the Nystrom approximation based spectral clustering algorithm was also set to $2,000$. We first observe that the RFF clustering algorithm took longer than the SV clustering algorithm on all the data sets. Though both algorithms require the computation of the data matrix H, the time taken to perform this computation was insignificant when compared to the *k*-means clustering time. RFF clustering involves running *k*-means on a $2m$-dimensional matrix, which takes longer than running *k*-means on a C-dimensional matrix. Although the SV clustering algorithm includes computing the singular vectors of H, the overhead of performing SVD is small, rendering it more efficient than the RFF clustering algorithm. On the CIFAR-10 data set, the SV clustering algorithm was at least 15 times

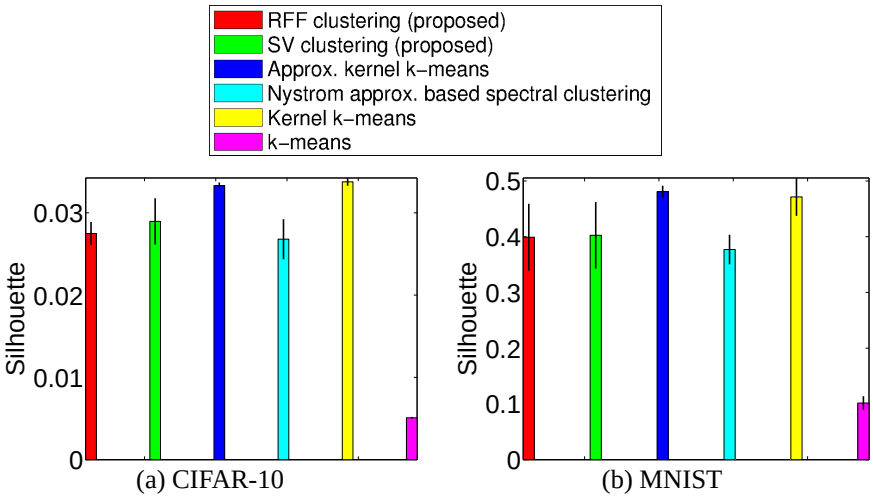
faster than the RFF clustering algorithm. On the MNIST data set, the SV clustering algorithm was about 20 times faster than the RFF clustering algorithm. Similar speedups were obtained for the other data sets as well. We will see later that the SV clustering algorithm achieves similar clustering accuracy as the RFF clustering algorithm. So we conclude that the SV clustering algorithm is more suitable for large scale kernel clustering than the RFF clustering algorithm.

The Nystrom approximation based spectral clustering algorithm finds the clusters by executing *k*-means on the top eigenvectors of a low rank approximate kernel matrix derived from a randomly sampled data subset of size m. It first obtains the eigenvectors of an $m \times m$ matrix and then extrapolates them to the top eigenvectors of the $n \times n$ kernel matrix. As the SV clustering algorithm only finds the top singular vectors of an $s \times m$ matrix, it is more efficient than the Nystrom approximation based spectral clustering algorithm. The SV clustering algorithm was also faster than approximate kernel *k*-means on all the data sets.

As expected, the SV algorithm was faster than the kernel *k*-means algorithm on the CIFAR-10 and MNIST data sets. As it is prohibitive to execute kernel *k*-means on the large Forest Cover Type, Imagenet-34, Poker and Network Intrusion data sets, we randomly selected a subset of $50,000$ points from these data sets, executed kernel *k*-means on this subset to obtain the cluster centers, and assigned the remaining points to the closest center. We recorded the time taken for this procedure as the time taken by kernel *k*-means on these large data sets. The SV algorithm was faster than this approximate version of kernel *k*-means as well on all the data sets. When the dimensionality of the data set was greater than the number of clusters in it, the SV clustering algorithm ran faster than the *k*-means algorithm.

3.5.4.2 Cluster quality

Figure 3.2 records the silhouette coefficient values of the proposed and baseline algorithms on the CIFAR-10 and MNIST data sets. The proposed algorithms achieved values comparable to the kernel *k*-means algorithm and the approximate kernel *k*-means algorithm, showing that they yield



(a) CIFAR-10 (b) MNIST

Figure 3.2 Silhouette coefficient values of the partitions obtained using the RFF and SV clustering algorithms. The parameter m, which represents the number of Fourier components for the RFF and SV clustering algorithms, and the sample size for the approximate kernel *k*-means and Nystrom approximation based spectral clustering algorithms, is set to $m = 2,000$.

similar partitions. The silhouette coefficient values of the Nystrom approximation based spectral clustering algorithm were marginally lower than those of the remaining kernel-based clustering algorithms. The *k*-means algorithm yielded non-compact partitions with silhouette values closer to 0.

Figure 3.3 shows the NMI values achieved by the proposed algorithms and the baseline algorithms. We first observe that the accuracy of all the kernel-based algorithms, including the proposed algorithms, was better than that of the *k*-means algorithm, demonstrating the fact that incorporating a non-linear similarity function improves the clustering performance. On the CIFAR-10 and MNIST data set, we observed that the performance of both our algorithms was similar to that of kernel *k*-means. Comparison with kernel *k*-means is not feasible on the remaining data sets due to their large size. The proposed algorithms outperformed the approximate version of kernel *k*-means in which a subset of the data was clustered and the remaining points were assigned to the closest center. The proposed algorithms' performance was significantly better than that of the Nystrom approximation based spectral clustering algorithm on all data sets. They performed only marginally

worse than the approximate kernel *k*-means algorithm. The difference in the NMI values of the RFF clustering algorithm and the SV clustering is minimal for most data sets.

3.5.4.3 Parameter sensitivity

The number of Fourier components m plays a crucial role in the performance of the RFF and SV clustering algorithms. The running time of the algorithms is compared with the approximate kernel *k*-means and the Nystrom spectral clustering algorithms for different values of m in Table 3.3. In the table, m represents the number of Fourier components in the context of the RFF and SV clustering methods, and it represents the size of the sample drawn from the data set in the context of approximate kernel *k*-means and Nystrom approximation based spectral clustering. As observed earlier, the SV algorithm is faster than the RFF clustering algorithm. For instance, the SV algorithm is about 15 times faster than the RFF algorithm on the CIFAR-10 data set when $m = 100$. The speedup factor increased as the number of Fourier components m increases. We note that the speedup on the Network Intrusion data set became significant only when $m \geq 500$. The SV clustering algorithm was also faster than approximate kernel *k*-means for all values of m, due to the fact that unlike the approximate kernel *k*-means algorithm, the dimensionality of the input to the *k*-means step (which dominates the running time) remains constant despite the increase in m. The dimensionality of the input kernel in the approximate kernel *k*-means algorithm increases linearly with m.

The silhouette coefficient values achieved by the algorithms on the CIFAR-10 and MNIST data sets, for different values of m are shown in Figure 3.4. We first observe that the silhouette values achieved by the proposed RFF and SV clustering algorithms increased significantly as m increased. The values were initially much lower than those achieved by the approximate kernel *k*-means and Nystrom approximation based spectral clustering algorithms, but became comparable when $m \geq 1,000$.

The NMI values achieved by the algorithms for different values of m are shown in Figure 3.5.

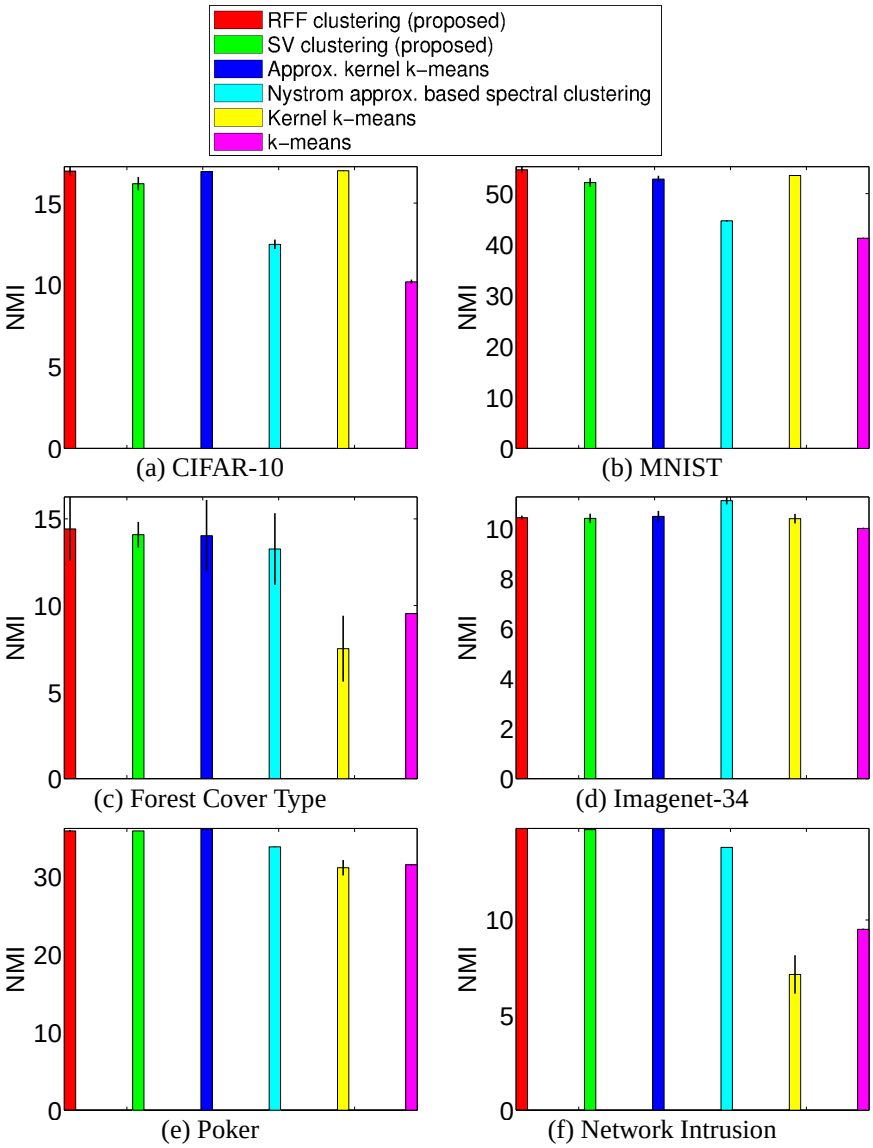
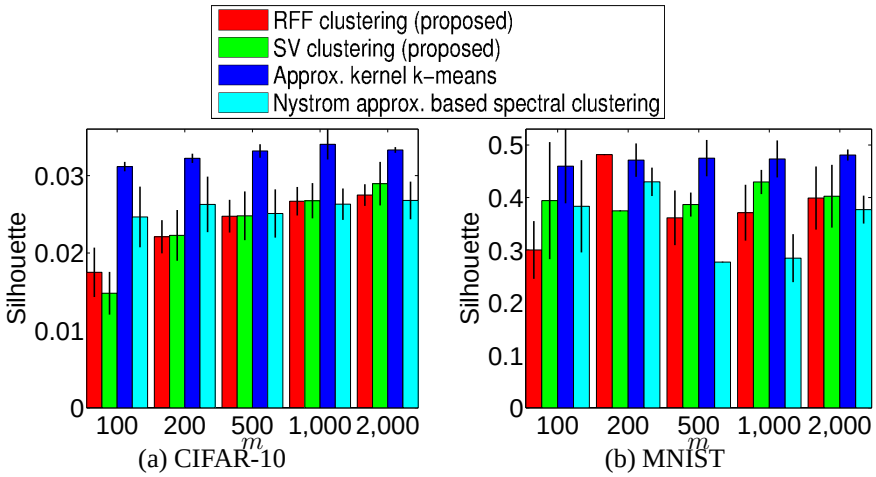


Figure 3.3 NMI values (in %) of the partitions obtained using the RFF and SV clustering algorithms, with respect to the true class labels. The parameter m, which represents the number of Fourier components for the RFF and SV clustering algorithms, and the sample size for the approximate kernel *k*-means and Nystrom approximation based spectral clustering algorithms, is set to $m = 2,000$. It is not feasible to execute kernel *k*-means on the large Forest Cover Type, Imagenet-34, Poker, and Network Intrusion data sets due to their large size. The approximate NMI values of kernel *k*-means on these data sets are obtained by first executing kernel *k*-means on a randomly chosen subset of $50,000$ data points to find the cluster centers, and then assigning the remaining points to the closest cluster center.

We note that, although the SV clustering algorithm performed worse than the RFF clustering algorithm in terms of NMI when m is small, it yielded similar performance as the RFF clustering algorithm when m was substantially large. On the MNIST data set, as the value of m increased from 100 to $2,000$, the average NMI achieved by the RFF clustering algorithm increased by about 15% whereas the SV clustering algorithm achieved an increase of 20%. Similar rates of increase were observed on other data sets also. This verifies our claim that the approximation error of the SV clustering algorithm decreases at a higher rate with respect to the parameter m, than that of the RFF clustering algorithm. While the NMI values of the SV clustering method are higher than those of the Nystrom spectral clustering method for all m values on most data sets, they are only marginally lower than those of the approximate kernel *k*-means algorithm for small m, and become close to the approximate kernel *k*-means values as m increases.



(a) CIFAR-10 (b) MNIST

Figure 3.4 Effect of the number of Fourier components m on the silhouette coefficient values of the partitions obtained using the RFF and SV clustering algorithms. Parameter m represents the number of Fourier components for the RFF and SV clustering algorithms, and the sample size for the approximate kernel *k*-means and Nystrom approximation based spectral clustering algorithms.

3.5.4.4 Scalability

We analyze the scalability of the proposed RFF and SV clustering algorithms for different values of n, d, C using the synthetic concentric circles data set. We set the number of Fourier features

Table 3.3 Effect of the number of Fourier components m on the running time (in seconds) of the RFF and SV clustering algorithms on the six benchmark data sets. Parameter m represents the number of Fourier components for the RFF and SV clustering algorithms, and the sample size for the approximate kernel *k*-means and Nystrom approximation based spectral clustering algorithms.

m	**RFF clustering (proposed)**	**SV clustering (proposed)**	**Approx. kernel *k*-means**	**Nystrom approx. based based spectral clustering**
100	89.94 (± 18.96)	5.39 (± 1.63)	12.29 (± 4.66)	0.91 (± 0.16)
200	176.47 (± 47.59)	6.09 (± 1.76)	39.91 (± 15.11)	1.86 (± 0.20)
500	449.23 (± 103.61)	10.71 (± 3.32)	13.20 (± 2.14)	5.61 (± 1.89)
1,000	1,176.74 (± 276.07)	16.46 (± 6.54)	49.50 (± 22.17)	26.24 (± 5.26)
2,000	3,418.21 (± 907.14)	58.32 (± 38.68)	37.01 (± 6.52)	116.13 (± 1.97)

(a) CIFAR-10

m	**RFF clustering (proposed)**	**SV clustering (proposed)**	**Approx. kernel *k*-means**	**Nystrom approx. based based spectral clustering**
100	85.36 (± 25.64)	3.85 (± 2.37)	26.57 (± 3.12)	6.00 (± 0.89)
200	122.31 (± 48.31)	4.66 (± 1.78)	17.98 (± 7.99)	46.70 (± 8.51)
500	272.57 (± 111.25)	9.22 (± 1.22)	24.72 (± 8.46)	342.38 (± 105.80)
1,000	517.48 (± 44.6)	17.46 (± 1.43)	36.34 (± 6.92)	914.18 (± 215.77)
2,000	1,089.26 (± 483.63)	39.94 (± 5.64)	86.43 (± 12.71)	4163.76 (± 383.37)

(b) MNIST

Table 3.3 (cont'd)

$\mathbf{m}$	**RFF clustering (proposed)**	**SV clustering (proposed)**	**Approx. kernel *k*-means**	**Nystrom approx. based based spectral clustering**
100	154.97 (± 65.72)	9.62 (± 2.57)	19.10 (± 6.35)	11.75 (± 1.73)
200	174.88 (± 65.36)	10.77 (± 1.67)	24.21 (± 12.48)	13.65 (± 1.59)
500	534.01 (± 216.18)	22.15 (± 6.08)	32.48 (± 11.64)	41.92 (± 7.89)
1,000	1,032.58 (± 221.56)	35.46 (± 5.20)	66.15 (± 19.25)	124.83 (± 38.32)
2,000	2,078.63 (± 617.22)	76.99 (± 17.04)	157.48 (± 27.37)	534.77 (± 323.76)

(c) Forest Cover Type

$\mathbf{m}$	**RFF clustering (proposed)**	**SV clustering (proposed)**	**Approx. kernel *k*-means**	**Nystrom approx. based based spectral clustering**
100	24.43 (± 0.92)	17.72 (± 1.09)	551.70 (± 120.53)	125.82 (± 8.26)
200	57.66 (± 2.15)	33.82 (± 0.96)	676.39 (± 10.94)	183.31 (± 4.63)
500	163.74 (± 5.54)	84.34 (± 4.62)	906.07 (± 209.53)	461.52 (± 7.48)
1,000	340.23 (± 11.30)	160.89 (± 5.65)	1028.99 (± 34.83)	586.66 (± 91.72)
2,000	1,333.85 (± 6.53)	212.32 (± 4.75)	1261.02 (± 37.39)	1841.47 (± 123.82)

(d) Imagenet-34

Table 3.3 (cont'd)

$\mathbf{m}$	**RFF clustering (proposed)**	**SV clustering (proposed)**	**Approx. kernel *k*-means**	**Nystrom approx. based based spectral clustering**
100	144.22 (± 11.88)	12.32 (± 1.70)	55.57 (± 11.22)	10.88 (± 1.65)
200	411.32 (± 34.34)	17.35 (± 2.07)	89.14 (± 32.12)	46.78 (± 4.21)
500	654.98 (± 132.70)	22.82 (± 2.48)	117.57 (± 20.17)	90.57 (± 18.57)
1,000	2,287.53 (± 159.06)	27.37 (± 2.09)	202.84 (± 44.25)	261.14 (± 20.51)
2,000	4,530.44 (± 276.37)	41.08 (± 2.57)	256.26 (± 44.84)	479.73 (± 47.46)

(e) Poker

$\mathbf{m}$	**RFF clustering (proposed)**	**SV clustering (proposed)**	**Approx. kernel *k*-means**	**Nystrom approx. based based spectral clustering**
100	2,252.44 (± 465.94)	147.93 (± 62.03)	736.59 (± 238.31)	145.21 (± 22.76)
200	5,371.85 ($\pm 1,765.02$)	258.86 (± 41.32)	697.04 (± 442.25)	169.27 (± 38.15)
500	5,296.87 ($\pm 3,321.66$)	245.37 (± 158.57)	586.14 (± 130.23)	366.42 (± 175.57)
1,000	24,151.47 ($\pm 6,351.34$)	435.53 (± 189.07)	763.75 (± 88.55)	589.57 (± 54.14)

(f) Network Intrusion

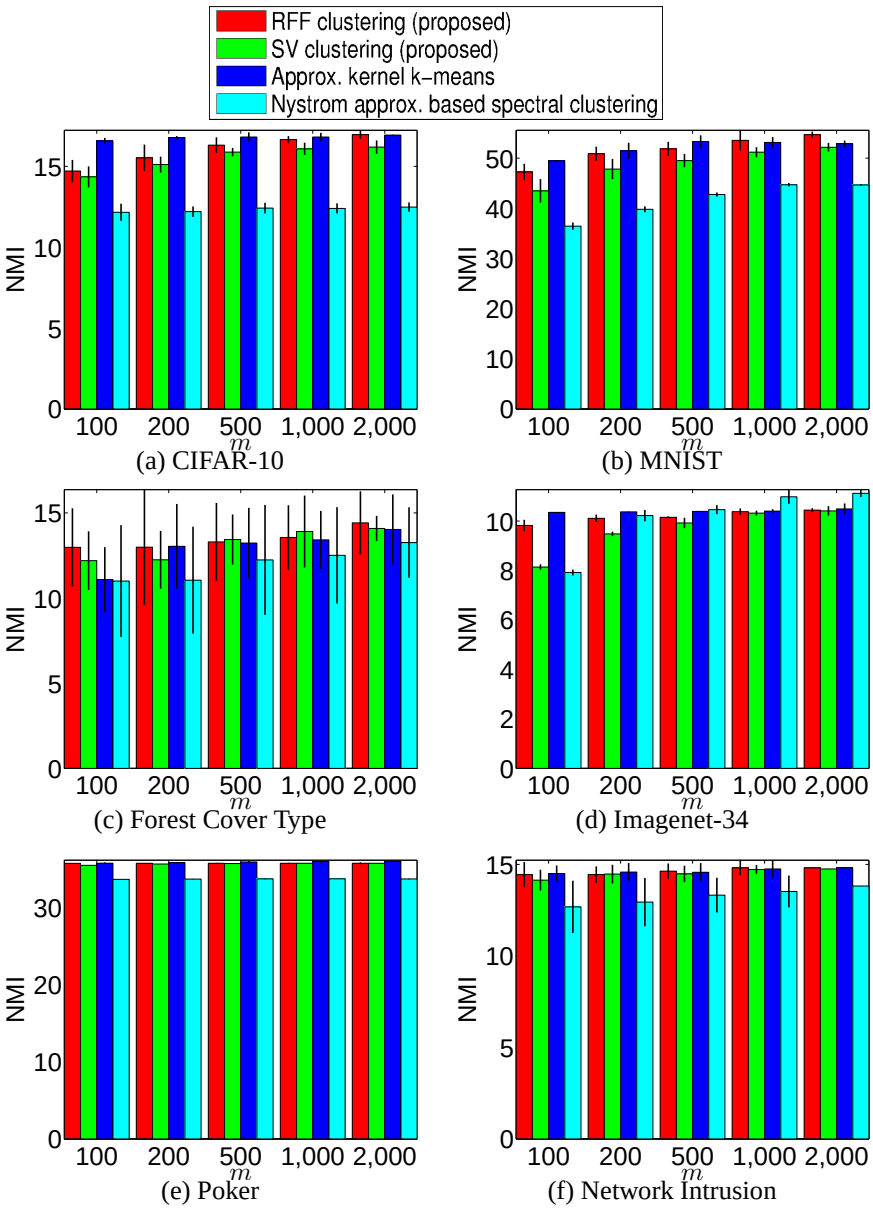


Figure 3.5 Effect of the number of Fourier components m on the NMI values (in %) of the partitions obtained using the RFF and SV clustering algorithms, on the six benchmark data sets. Parameter m represents the number of Fourier components for the RFF and SV clustering algorithms, and the sample size for the approximate kernel *k*-means and Nystrom approximation based spectral clustering algorithms.

m to $1,000$. Figures 3.6(a) and 3.7(a) show that the running time of the RFF and SV clustering algorithms vary nearly linearly as the number of points in the data set varies from $n = 100$ to $n = 10^7$, with dimensionality $d = 100$ and number of clusters $C = 10$. The scalability plots of the RFF and SV clustering algorithms are similar to the scalability plots of the approximate kernel *k*-means algorithm, because all three algorithms have linear time complexity with respect to n.

The dimensionality of the data set affects the time taken for calculation of the Fourier features. The order of increase in the running time of the two algorithms as d varies from $d = 10$ to $d = 1,000$, with $n = 10^6$ and $C = 10$, are shown in Figures 3.6(b) and 3.7(b).

As the number of clusters was increased from $C = 10$ to $C = 1,000$, with $n = 10^5$ and $d = 100$, the running time of the RFF and SV algorithms increases almost linearly with C, as shown in Figures 3.6(c) and 3.7(c). We note that the number of clusters affects the running time of the SV clustering algorithm more than that of the RFF clustering algorithm, because the SV clustering algorithm projects the data into a C-dimensional space before clustering.

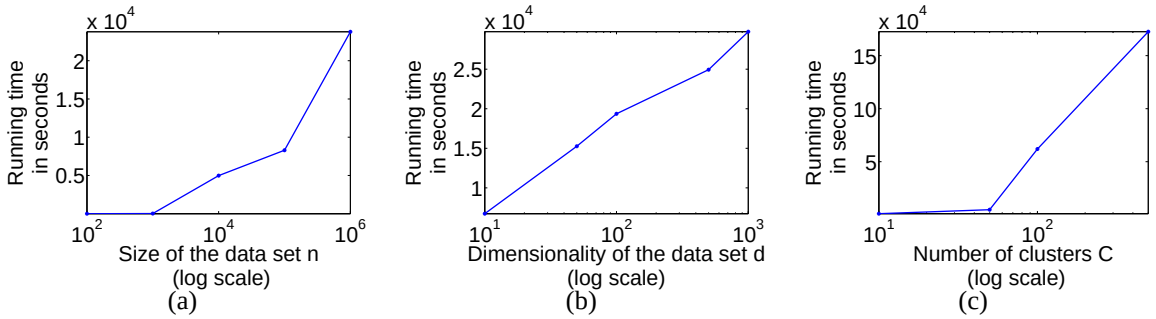


Figure 3.6 Running time of the RFF clustering algorithm for different values of (a) n, (b) d and (c) C.

3.5.4.5 Out-of-sample clustering

To evaluate the performance of our algorithm on out-of-sample data points, we divided each data set into two parts, one containing 80% of the data, and the other containing the remaining 20%. We call the first part as the *training set* and the second part as the *test set*, in accordance with

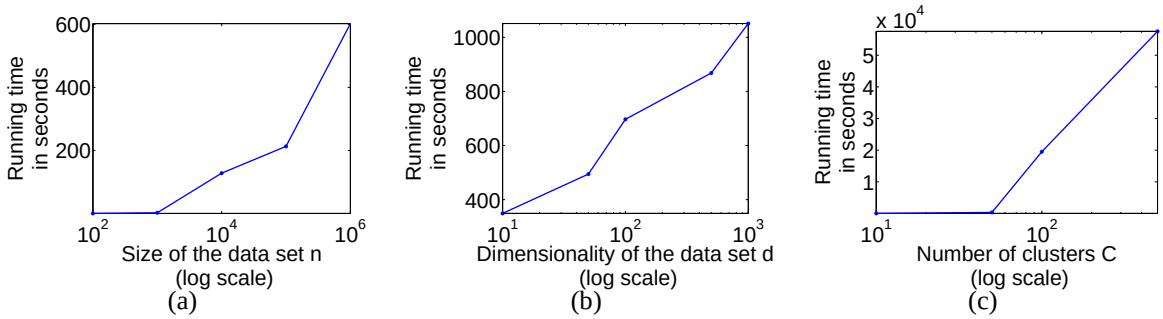


Figure 3.7 Running time of the SV clustering algorithm for different values of (a) n, (b) d and (c) C.

the convention followed in supervised learning problems. We computed the cluster centers using the training set, and assigned each test point to the closest cluster center, using the SV clustering algorithm. The class assignment of a test point was determined by the majority class in the cluster to which it was assigned.

We compared the performance of our algorithm with the weighted kernel principal component analysis (WKPCA) extension for out-of-sample data points, proposed in [11]. This method first finds the eigenvectors $\mathcal{Z} = (\mathbf{z}_1, \ldots, \mathbf{z}_C)$ of the matrix $D^{-1}MK$ corresponding to its smallest C eigenvalues, where $D = \text{diag}\left(K^\top \mathbf{1}\right)$ is the degree matrix and $M = I - \frac{1}{\mathbf{1}^\top D^{-1}\mathbf{1}} \mathbf{1}\mathbf{1}^\top D^{-1}$ is a centering matrix, and then encodes the eigenvectors into binary codewords based on their sign. These codewords are clustered to obtain C binary codewords $\{\mathbf{c}_1, \ldots, \mathbf{c}_C\}$. The following procedure is employed to obtain the cluster label for a new point $\mathbf{x}^*$:

(i) Project $\mathbf{x}^*$ on to the space spanned by the eigenvectors of the training set as $\varphi^* \mathcal{Z}$, where $\varphi^* = \left(\kappa\left(\mathbf{x}^*, \mathbf{x}_1\right), \ldots, \kappa\left(\mathbf{x}^*, \mathbf{x}_n\right)\right)$.

(ii) Compute the codeword $\mathbf{c}^* = \text{sign}(\varphi^*)$.

(iii) Assign $\mathbf{x}^*$ to the cluster k which minimizes $\mathbf{d}_{HM}\left(\mathbf{c}^*, \mathbf{c}_k\right)$, where $\mathbf{d}_{HM}$ represents the Hamming distance [66] between the vectors $\mathbf{c}^*$ and $\mathbf{c}_k$, defined as

$$\mathbf{d}_{HM}(\mathbf{x}_a, \mathbf{x}_b) = |\mathbf{x}_a - \mathbf{x}_b|$$

The WKPCA extension requires the eigendecomposition of an $n \times n$ matrix, which takes $O(n^3)$ time. In addition, an $O(n)$ vector needs to be computed to perform label assignment.

We also compare the performance of the proposed algorithm with the approximate kernel *k*-means algorithm. The test point $\mathbf{x}^*$ is added to the cluster whose center, given by

$$\mathbf{c}_k(\cdot) = \alpha_k^\top \widehat{K} \alpha_k - 2\varphi^\top \alpha_k,$$

is closest. In the above expression, $\varphi = [\kappa(\mathbf{x}^*, \widehat{\mathbf{x}}_1), \ldots, \kappa(\mathbf{x}^*, \widehat{\mathbf{x}}_m)]$, $\{\widehat{\mathbf{x}}_1, \ldots, \widehat{\mathbf{x}}_m\}$ are the set of sampled data points, $\widehat{K}$ is the kernel similarity between the sampled points, and α_k is the k^{th} row of cluster center coefficient matrix α, given by (2.10).

We report the running time and accuracy on the six data sets in Table 3.4. The running time is divided into training time and testing time. The training time for WKPCA includes the time taken to compute the kernel matrix for the training data and its eigenvectors, and the time taken to convert the eigenvectors to the cluster codewords. The testing time is the time taken for data projection and Hamming distance computation for all the test data points. For the approximate kernel *k*-means algorithm, the training time includes the time to cluster the training data and obtain the cluster center coefficient matrix α. The testing time includes the time taken to compute the similarity between the test data points and the sampled data points, and the time to assign the cluster labels to the test data points. For SV clustering, the training time is defined as the time taken to compute the random fourier features and the singular vectors for the training data, and the testing time is defined as the time taken to assign labels to test data.

The WKPCA method took about 40 seconds, on an average, to assign labels to the $12,000$ test images in the CIFAR-10 data set, whereas our method took less than 5 seconds, for $m = 1,000$. On the MNIST data set, the WKPCA method took about 940 seconds to cluster the test set containing $14,000$ data points, significantly longer than the proposed algorithm, which took around 60 seconds, for $m = 1,000$. It is infeasible to evaluate the performance of WKPCA on the

large data sets. We observed that both the proposed algorithm and the WKPCA method achieved similar classification performance on the CIFAR-10 and MNIST data sets. A reasonably good accuracy was achieved on the remaining large data sets also. The proposed algorithm also runs faster than the approximate kernel *k*-means algorithm, and achieves comparable test accuracy.

Table 3.4 Running time (in seconds) and prediction accuracy (in %) for out-of-sample data points. Parameter m represents the sample size for the approximate kernel *k*-means algorithm and the number of Fourier components for the SV clustering algorithm. The value of m is set to $1,000$ for both the algorithms. It is not feasible to execute the WKPCA algorithm on the large Forest Cover Type, Imagenet-34, Poker, and Network Intrusion data sets due to their large size.

Data set		**CIFAR -10**	**MNIST**	**Forest Cover Type**	**Imagenet -34**	**Poker**	**Network Intrusion**
Training time	**WKPCA**	755.02 ($\pm$91.35)	910.90 ($\pm$84.37)	-	-	-	-
	Approx. kernel *k*-means	26.24 ($\pm$2.36)	62.11 ($\pm$3.58)	39.38 ($\pm$3.93)	1913 ($\pm$414)	391.04 ($\pm$120.1)	998.36 ($\pm$812.73)
	SV clustering	5.96 ($\pm$0.83)	10.48 ($\pm$0.51)	25.28 ($\pm$1.61)	155.89 ($\pm$4.77)	49.75 ($\pm$6.09)	115.73 ($\pm$3.50)
Testing time	**WKPCA**	39.68 ($\pm$2.77)	29.50 ($\pm$4.69)	-	-	-	-
	Approx. kernel *k*-means	22.47 ($\pm$2.05)	55.38 ($\pm$1.75)	26.76 ($\pm$0.97)	1543 ($\pm$412)	373.45 ($\pm$119.5)	213.68 ($\pm$29.28)
	SV clustering	5.33 ($\pm$2.25)	2.12 ($\pm$0.57)	5.97 ($\pm$2.33)	80.24 ($\pm$0.02)	14.24 ($\pm$0.51)	121.35 ($\pm$32.50)
Accuracy	**WKPCA**	80.70	84.84	-	-	-	-
	Approx. kernel *k*-means	83.08 ($\pm$0.01)	88.76 ($\pm$0.001)	59.39 ($\pm$0.10)	88.50 ($\pm$0.002)	55.40 ($\pm$0.001)	57.30 ($\pm$0.03)
	SV clustering	83.13 ($\pm$0.04)	88.33 ($\pm$0.52)	58.42 ($\pm$0.64)	80.56 ($\pm$0.01)	55.41 ($\pm$0.04)	59.03 ($\pm$0.03)

3.6 Summary

The RFF clustering and the SV clustering algorithms, proposed in this chapter, use random Fourier features to obtain a good approximation of kernel clustering using an efficient linear clustering algorithm. We have analytically bound the approximation error of both these methods. We have shown that, when there is a large gap in the eigenspectrum of the kernel matrix, as is the case in most big data sets, the SV clustering algorithm which clusters the singular vectors of the random Fourier features is a more effective and scalable approximation of kernel clustering, allowing large data sets with millions of data points to be clustered using kernel-based clustering. It also solves the out-of-sample clustering problem efficiently. The RFF clustering algorithm can be trivially parallelized by replicating the random Gaussian matrix across the computing nodes, calculating the random Fourier features for a subset of the data in each node, and employing the parallel *k*-means algorithm to cluster the random Fourier feature matrix, to obtain the cluster labels. The SV clustering algorithm can be similarly parallelized, by using the distributed Lanczos eigensolver to obtain the eigenvectors of the random Fourier feature matrix.

The approximate kernel *k*-means algorithm in Chapter 2 and the random Fourier features-based algorithms in this chapter are all based on sampling the data set and using the samples as basis functions for the cluster centers. While approximate kernel *k*-means employs the data-dependent Nystrom kernel approximation, and obtains the basis functions by factorizing the kernel matrix, the basis functions in RFF and SV clustering algorithms are dependent on the kernel function. Therefore, these algorithms require a large number of Fourier components to achieve cluster quality equivalent to that of the approximate kernel *k*-means algorithm. Kernel selection is also very crucial in the RFF and SV clustering algorithms. We have focused on using scale-invariant kernel functions in our work, but these algorithms can be extended to polynomial and intersection kernels using the schemes prescribed in [92] and references therein, to obtain the basis functions.

Chapter 4

Stream Clustering

4.1 Introduction

In addition to the large volume, big data is also characterized by "velocity" - the continuous pace at which data flows in from sources such as sensors, machines, networks, and user interaction with websites. Analysis of this real-time data can help in making valuable decisions. For instance, intrusions can be detected in IP networks by analyzing the network traffic.

Clustering streaming data is challenging due to the following two reasons:

(i) Streaming data sets are often too large to load in memory; they could potentially be unbounded. Only a small subset of the data may be stored, depending on the amount of memory available. So the data can be accessed at most once, and

(ii) the data is non-stationary, i.e. the distribution of the data changes over time. The data that arrived more recently has higher relevance than the older data.

Batch clustering algorithms such as *k*-means and kernel *k*-means, assume that the data is completely available in memory at the time of clustering. They also assume that the input data is drawn from the mixture of a fixed set of distributions, and the aim of clustering is to identify these

component distributions. Therefore, batch clustering algorithms cannot be directly used to cluster streaming data. Stream clustering algorithms model the data dynamically. Cluster labels are assigned to data points as they arrive, in an online manner. Stream clustering algorithms generally consist of two stages: (i) an online phase, where the stream data is summarized into "prototypes" as it arrives, and (ii) an offline phase where these prototypes are used to obtain the clusters. The set of prototypes are dynamically updated to account for the evolution of the clusters in the streaming data.

Many stream clustering algorithms use measures such as the Euclidean distance to define the pairwise similarity. As demonstrated in the earlier chapters, kernel-based algorithms achieve better clustering quality than linear clustering algorithms. However, kernel-based clustering algorithms are ill-suited to streams because of their high computational complexity. In this chapter, we adapt the kernel *k*-means algorithm to efficiently handle streaming data. The proposed algorithm samples the data points as they arrive and constructs an approximate kernel matrix using the sampled points. The sampling is performed with probability proportional to the statistical leverage scores [34] of this matrix, a measure of the importance of the data points. The sampled data points are stored in memory and used to determine the cluster labels of the incoming data points. We show that only a small subset of the data needs to be stored in memory, thereby enhancing the efficiency of kernel clustering for data streams.

4.2 Background

Data stream clustering has been studied extensively in the pattern recognition and data mining literature. Most stream clustering algorithms summarize the data stream using special data structures, and obtain the cluster representatives using this summary. They differ by the data structures used to summarize the data; common data structures are trees, coresets, and grids (See Table 4.1).

Stream and LSearch algorithms split the incoming data into chunks, cluster the chunks indi-

Table 4.1 Major published approaches to stream clustering.

Approaches for stream clustering	**Examples**
CF-Trees	Stream [79], Stream LSearch [140], Scalable *k*-means [30], Single-pass *k*-means [62]
Microcluster trees	CluStream [8], ClusTree [98], ClusTrel [124], DenStream [32], HPStream [9]
Coresets	StreamKM++ [6]
Grids	D-Stream [36], ODAC [149]
Approximate clustering	Streaming *k*-means approximation [10], Fast streaming *k*-means [162]
Kernel-based	Incremental spectral clustering [139], Adaptive non-linear clustering [86], sKKM [84], TechnoStream [134]

vidually to find the cluster prototypes, and then cluster these prototypes to obtain the final clusters [79, 140]. These algorithms cannot be used to perform real-time clustering. The Clustering Feature (CF) Tree was introduced by Zhang *et al.* as a part of the BIRCH clustering algorithm [197]. A CF-Tree summarizes the data stream into a hierarchy of nodes. Each node contains a set of CF-vectors comprising the linear sum and the squared sum of a set of points, which are close to each other. The CF-Tree has been used in several stream clustering algorithms such as scalable *k*-means, and single-pass *k*-means algorithms [30, 62]. The idea of CF-vectors was then extended to "micro-clusters" which include the temporal information about the data [32, 98, 124]. This information is used to detect evolutionary changes in the data stream. For instance, the CluStream algorithm stores the linear and squared sums of the timestamps of the data points in the microcluster, in addition to the linear sum and the squared sum of the data points. These timestamp values are used to assign weights to the data points, thereby giving more importance to the new data than older data while clustering. Similarly, the HPStream algorithm weights the clusters using the temporal information and assigns data to more recent clusters [9].

A coreset is a weighted subset of points that approximate the input data set up to a pre-defined error margin. The StreamKM++ algorithm summarizes the data stream into a set of coresets or-

ganized into a hierarchy known as the coreset tree [6]. Each node in the tree contains a subset of points represented by a set of prototypes. The final clusters are obtained by grouping the coreset representatives in the root node of the coreset tree. Grid-based algorithms such as DStream and DGClust partition the d-dimensional feature space into grid cells [36,149]. Each cell is represented by a tuple containing the timestamps, a cluster label and the density of the grid. Data points are added to the grids and the grid summaries are updated incrementally, as the data points arrive. Approximate clustering algorithms such as streaming *k*-means [10, 162] choose a subset of the points from the stream, ensuring that the selected points are as distant from each other as possible, and execute *k*-means on the data subset.

To the best of our knowledge, based on published literature, very few attempts have been made to use non-linear similarity measures for clustering data streams. The agglomerative hierarchical clustering algorithm is adapted to use kernel distance measures in [193]. The incremental spectral clustering algorithm [139] extends spectral clustering to stream data by treating each new edge in the graph as a vector appended to the similarity matrix. The graph Laplacian, its eigenvalues and eigenvectors are updated incrementally with the new edges.

The stream kernel *k*-means algorithm [84] divides the data set into windows of fixed time-steps, and performs clustering using the data points in every two consecutive windows. Information from the current time-step is passed on to the to the next time-step in the form of meta-vectors containing weights for each of the C clusters. Jain *et al.* proposed a two-tier system called the adaptive non-linear clustering algorithm to perform stream clustering using non-linear similarity [86]. In the first tier, the incoming data points are partitioned into segments, separated from each other by novel data points. A data point $\mathbf{x}$ is considered novel if the kernel-based distance from $\mathbf{x}$ to the mean of the data points in the current segment is greater than the user-defined threshold. In the second tier, the representative segments are identified and projected into a low-dimensional space spanned by the dominant principal coordinates of the data in kernel space [76]. The cluster labels for the data points are obtained by clustering the low-dimensional representations of the data. This technique

requires the eigendecomposition of a large number of points in the second tier. The proposed method uses the complete history of data, and does not require complex operations, unlike the existing methods.

4.3 Approximate Kernel *k*-means for Streams

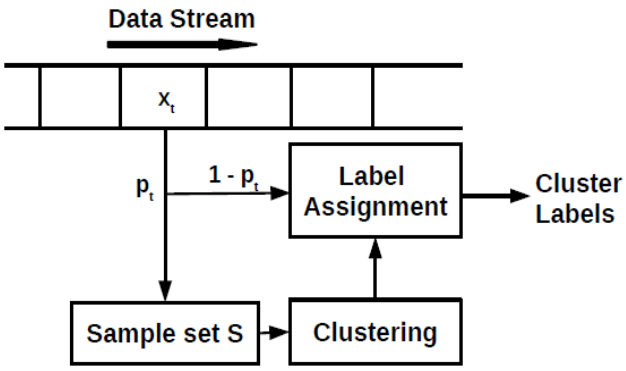


Figure 4.1 Schema of the proposed approximate stream kernel *k*-means algorithm.

In Chapter 2, we presented the approximate kernel *k*-means algorithm which constrained the cluster centers to the span of a subset of the data points. We employ a similar strategy to cluster streaming data. The key idea is to sample the data points as they arrive and construct the kernel matrix incrementally using the sampled points. This approximate kernel matrix is used to cluster the sampled points. The cluster labels are assigned to the unsampled data points using their kernel similarity with the sampled points. A high level overview of the proposed clustering framework is presented in Figure 4.1. Our framework consists of three primary components, working in tandem: (i) importance sampling, (ii) clustering, and (iii) cluster label assignment. The sampling component samples the points from the stream, and constructs the approximate kernel matrix. The clustering and label assignment components update the clusters and the number of clusters dynamically, and assign cluster labels to all the data points in the stream.

We describe each of these components in the following sections:

4.3.1 Sampling

One of the obstacles to using kernel *k*-means for clustering stream data is that it requires the computation of the $n \times n$ kernel matrix, where n is the number of points in the data set. It is infeasible to compute the full kernel matrix for stream data because n is potentially unbounded. The approximate kernel-based clustering algorithms proposed in Chapters 2 and 3 also need to store the entire data in memory, before constructing the approximate kernel matrices. The stream clustering algorithm proposed in this chapter alleviates this issue by incrementally sampling a subset of the points from the stream, and using only this subset to construct the kernel matrix. We maintain a buffer S in memory to store the sampled points; the number of points s in S is constrained by the user-defined parameters m and M ($m \leq s \leq M$). Let K_{t-1} represent the kernel matrix at time $(t-1)$ with $K_1 = \kappa(\mathbf{x}_1, \mathbf{x}_1)$. When a data point $\mathbf{x}_t$ arrives at time t, we update the kernel matrix as

$$K_t = \begin{cases} \begin{bmatrix} K_{t-1} & \varphi^\top \\ \varphi & \kappa(\mathbf{x}_t, \mathbf{x}_t) \end{bmatrix} & \text{with probability } p_t, \\ K_{t-1} & \text{with probability } 1 - p_t, \end{cases} \tag{4.1}$$

where $K_{t-1} = [\kappa(\mathbf{x}_i, \mathbf{x}_j)], \mathbf{x}_i, \mathbf{x}_j \in S$, and $\varphi = (\kappa(\mathbf{x}_t, \mathbf{x}_1), \ldots, \kappa(\mathbf{x}_t, \mathbf{x}_s))^\top$.

The simplest method of determining whether or not to add a data point $\mathbf{x}_t$ to S, is to perform independent Bernoulli trials, i.e. $\mathbf{x}_t$ is stored in S with probability $p_t = \frac{1}{2}$. However, Bernoulli sampling results in a large kernel approximation error, and requires a large number of points to be stored in memory[1]. To alleviate this issue, we perform importance sampling instead of Bernoulli sampling. The sampling probability p_t for each point $\mathbf{x}_t$ is based on its "impor-

[1]We demonstrate this using a synthetic data set in Figure 4.2, and using four large benchmark data sets in Section 4.5.

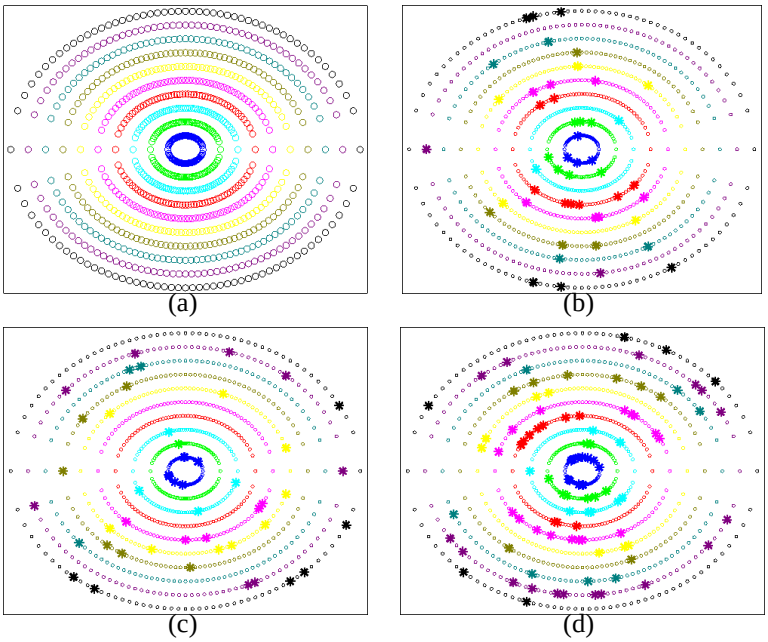


Figure 4.2 Illustration of importance sampling on a two-dimensional synthetic data set containing $1,000$ points along 10 concentric circles (100 points in each cluster), represented by "o" in Figure (a). Figure (b) shows 50 points sampled using importance sampling, and Figures (c) and (d) show 50 and 100 points selected using Bernoulli sampling, respectively. The sampled points are represented using "*". All the 10 clusters are well-represented by just 50 points sampled using importance sampling. On the other hand, 50 points sampled using Bernoulli sampling are not adequate to represent these 10 clusters (Cluster 4 in red has no representatives). At least 100 points are needed to represent all the clusters.

tance", defined in terms of the statistical leverage scores [56]. Let the kernel matrix K_t at time t be decomposed as $K_t \simeq V_C \Sigma_C V_C^\top$, where C represents the number of active clusters[2] at time t, $\Sigma_C = \text{diag}(\lambda_1, \ldots, \lambda_C)$ contains the highest C eigenvalues of K_t, and $V_C = (\mathbf{v}_1, \ldots, \mathbf{v}_C)$ contains the corresponding eigenvectors. The probability of adding point $\mathbf{x}_t$ to S is defined by

$$p_t = \frac{1}{C} \left|\left| V_C^{(t)} \right|\right|_2^2, \tag{4.2}$$

where $V_C^{(j)}$ is the j^{th} row of V_C. Statistical leverage scores measure the correlation between the eigenvectors of the matrix K_t and the standard basis. A high score indicates that the corresponding

[2]We refer to the set of clusters that the data points in the buffer S belong to at time t as the set of active clusters.

data point has a large influence in the approximation of the kernel matrix. The subset of data corresponding to the largest statistical leverage values are the most informative, and can represent the distribution of the entire data. By performing importance sampling on the data stream, the samples that have not been adequately represented by the existing samples are added to the buffer.

Statistical leverage scores have been used successfully to obtain low rank matrix approximations of large matrices, perform large scale regression and other large scale data analysis operations [28, 34]. The following lemma adapted from [74] shows that, at time t, the approximation error between the true kernel matrix for the t points $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t\}$ and the low-rank kernel matrix constructed using this sampling scheme is minimized, when the number of samples in S at time t is $s = \Omega(C \ln C)$:

Lemma 8. *Let K be a $t \times t$ SPSD matrix, and $V_C = (\mathbf{v}_1, \ldots, \mathbf{v}_C)$ represent the eigenvectors corresponding to the top C-dimensional eigenspace of K. Let K_B represent the $t \times s$ matrix obtained by sampling the columns of K with probability defined in* (4.2) *and $\widehat{K}$ be the $s \times s$ submatrix of K_B corresponding to the sampled columns. For a given failure probability $\delta \in (0, 1]$, and approximation factor $\epsilon \in (0, 1]$, if $s \geq 3200\epsilon^{-2} C \ln(4C/\delta)$, we have*

$$\left\| K - K_B \widehat{K}^{-1} K_B^\top \right\|_2 \leq \|K - K^*\|_2 + \epsilon^2 \|K - K^*\|_* ,$$

where K^* is the best C-rank approximation of K, and $\|\cdot\|_2$ and $\|\cdot\|_*$ represent the spectral norm and trace norm respectively[3].

By using importance sampling, we obtain a good approximation of the true kernel by sampling just a fraction of the data set. Figures 4.2(a)-(d) illustrate the advantage of importance sampling over Bernoulli sampling on a two-dimensional data set containing $1,000$ points from 10 clusters. Each true cluster is a concentric circle of varying radius, with 100 points, as shown in Figure 4.2(a).

[3]Lemma 8 bounds the error between the approximate kernel and the true kernel for a set of t data points. We demonstrate empirically in Section 4.5 that the accumulated error as time t increases is well-bounded.

Figure 4.2(b) also shows 50 points sampled using importance sampling. We observe that all the 10 clusters are adequately represented by the 50 sampled points. Figure 4.2(c) shows that 50 points sampled from the data using Bernoulli sampling do not represent all the clusters, as the probability of sampling data points from all the clusters is low. All the clusters are represented only when 100 points are sampled, as shown in Figure 4.2(d).

4.3.2 Clustering

Let s be the number of points in the buffer S and C be the number of active clusters[2] at time t. After the kernel matrix K_t is constructed in accordance with (4.1), the data points in S can be partitioned into C clusters by solving the kernel *k*-means problem

$$\max_{U \in \mathcal{P}} \text{tr}(\widetilde{U} K_t \widetilde{U}^\top), \tag{4.3}$$

where $U = (\mathbf{u}_1, \ldots, \mathbf{u}_C)^\top$ is the cluster membership matrix, $\widetilde{U} = \left[\text{diag}\,(U\mathbf{1})\right]^{-1/2} U$, domain $\mathcal{P} = \{U \in \{0,1\}^{C \times s} : U^\top \mathbf{1} = \mathbf{1}\}$, and $\mathbf{1}$ is a vector of all ones. The running time complexity of this step would be $O(s^2)$. We further reduce this complexity by constraining the cluster centers to a smaller subspace, spanning the top C eigenvectors of the kernel matrix K_t, along the lines of the spectral clustering algorithm. We pose the clustering problem as the following optimization problem:

$$\min_{U \in \mathcal{P}} \max_{\{\mathbf{c}_k(\cdot) \in \mathcal{H}_a\}_{k=1}^C} \sum_{k=1}^{C} \sum_{i=1}^{s} \frac{U_{k,i}}{s} \left|\left|\mathbf{c}_k(\cdot) - \kappa(\mathbf{x}_i, \cdot)\right|\right|^2_{\mathcal{H}_\kappa}, \tag{4.4}$$

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

$$\mathbf{c}_k(\cdot) = \sum_{i=1}^{s} \sum_{j=1}^{C} \frac{U_{k,i}}{n_k} \sqrt{\lambda_j} \mathbf{v}_{ij} = \frac{\mathbf{u}_k}{n_k} V_C \Sigma_C^{1/2},\ k \in [C], \tag{4.5}$$

where n_k is the number of points in the k^{th} cluster, and $\mathbf{u}_k = (U_{k,1}, U_{k,2}, \ldots, U_{k,s})^\top$. By substituting (4.5) in (4.4), we obtain the following trace maximization problem:

$$\max_{U \in \mathcal{P}} \text{tr}(\widetilde{U} V_C \Sigma_C V_C^\top \widetilde{U}^\top). \tag{4.6}$$

The above problem can be solved efficiently by executing *k*-means on the matrix $V_C \Sigma_C^{1/2}$. In the following lemma, we show that the error incurred due to the approximation (4.4) is bounded, when the lowest eigenvalues of the kernel matrix have small magnitudes, which is true for most real data sets [45]:

Lemma 9. *Let E and E_a represent the optimal clustering errors in* (4.3) *and* (4.6)*, respectively. We have*

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

Proof. Let $\{\mathbf{c}_k^*(\cdot)\}_{k=1}^C$ and U^* be the optimal solution to (4.3). Let $c_k^a(\cdot)$ represent the projection of c_k^* into the subspace $\mathcal{H}_a$. For any $\kappa(\mathbf{x}_i, \cdot)$, let $g_i(\cdot)$ and $h_i(\cdot)$ be the projections of $\kappa(\mathbf{x}_i, \cdot)$ into the subspace $\mathcal{H}_a$ and span$(\mathbf{v}_{C+1}, \ldots, \mathbf{v}_s)$, respectively. We have

$$\begin{aligned}
E_a &= \min_{U \in \mathcal{P}} \max_{\mathbf{c}_k(\cdot) \in \mathcal{H}_a} \sum_{k=1}^{C} \sum_{i=1}^{s} \frac{U_{k,i}}{s} ||\mathbf{c}_k(\cdot) - \kappa(\mathbf{x}_i, \cdot)||_{\mathcal{H}_\kappa}^2 \\
&\leq \sum_{k=1}^{C} \sum_{i=1}^{s} \frac{U_{k,i}^*}{s} ||\mathbf{c}_k^a(\cdot) - \kappa(\mathbf{x}_i, \cdot)||_{\mathcal{H}_\kappa}^2 \\
&\leq \sum_{k=1}^{C} \sum_{i=1}^{s} \frac{U_{k,i}^*}{s} \left(||c_k^a(\cdot) - g_i(\cdot)||_{\mathcal{H}_\kappa}^2 + ||h_i(\cdot)||_{\mathcal{H}_\kappa}^2 \right) \\
&\leq E + \frac{1}{s} \sum_{k=1}^{C} \sum_{i=1}^{s} ||h_i(\cdot)||_{\mathcal{H}_\kappa}^2 \leq E + \sum_{i=C+1}^{s} \lambda_i.
\end{aligned}$$

□

We note that the eigenvalues and eigenvectors do not need to be re-computed for clustering,

as they were already computed while calculating the leverage scores. This eliminates the need for computing and storing the kernel matrix K_t, as only its top eigenvalues and the corresponding eigenvectors are required for both sampling and clustering. Starting with $V_C = \mathbf{1}$ and $\Sigma_C = \kappa(\mathbf{x}_1, \mathbf{x}_1)$, we can update the eigensystem incrementally as the data points arrive. Efficient methods to update the eigenvectors and eigenvalues incrementally are discussed in Section 4.4.

4.3.3 Label Assignment

Data points are assigned cluster labels using the cluster centers obtained from the sampled data points in a manner similar to the SV clustering algorithm in Chapter 3, and the active clusters are updated using a fading cluster mechanism, similar to that used by the adaptive non-linear clustering algorithm [86]. Each cluster k is associated with a timestamp t_k representing the last time a data point was assigned the k^{th} cluster label, and a recency value defined by a monotonic function

$$f_k(t) = \exp\left(-\gamma\left(t - t_k\right)\right), \tag{4.7}$$

where γ is a user-defined parameter, representing the decay rate of a cluster [9]. A data point $\mathbf{x}_t$ is added to cluster k^* if

$$k^* = \arg\min_{k \in [C]} ||\mathbf{c}_k(\cdot) - g_t(\cdot)||^2_{\mathcal{H}_\kappa}, \text{ and } f_{k^*}(t) > \eta, \tag{4.8}$$

where $\mathbf{c}_k(\cdot)$ is the cluster center given by (4.5), $g_t(\cdot)$ is the projection of $\kappa(\mathbf{x}_t, \cdot)$ into the subspace spanned by the eigenvectors V_C, and η is a user-defined lifetime threshold which determines how long a cluster remains active. If the recency $f_{k^*}(t)$ of the closest cluster k^* is less than η, then a new cluster is created with the data point $\mathbf{x}_t$. After the cluster assignment is made, the timestamp and the recency value of the assigned cluster are updated. Clusters whose recency is less than η (called stale clusters) are deleted, and the data points in the buffer that belong to these stale clusters

are removed from the buffer.

Algorithm 8 describes the proposed stream clustering method. The input to the algorithm is the data stream $\mathcal{D}$, kernel function $\kappa(\cdot,\cdot)$, initial number of clusters C, buffer size parameters (m and M), and clustering fading mechanism parameters (γ and η). Selection of kernel function and initial number of clusters C is based on domain knowledge. Several articles in the literature describe techniques to learn the kernel function from the data [112, 177, 200]. The parameters m and M should be set such that the initial and final sample sets contain sufficient representatives from all the clusters. The parameters γ and η should be selected based on how fast the categories are expected to change in the stream. Heuristics to set these parameters are discussed further in Section 4.5.

4.4 Implementation and Complexity

The two major operations in the proposed algorithm are: computation of leverage scores, and clustering of the top C eigenvectors of the approximate kernel matrix using *k*-means. Both the operations require the eigenvalues and eigenvectors of the kernel matrix. Let s be the number of points in the sample set S at time t. Eigendecomposition of an $s \times s$ kernel matrix K_t takes $O(s^3)$ time, if performed naively. However, we can update the eigensystem incrementally using the fast rank-one update mechanism proposed in [31]. Given the eigendecomposition, $K_t = V\Sigma V^\top$, and vector $\varphi \in \Re^s$, this method finds the eigendecomposition of $\left(K_t + \varphi\varphi^\top\right)$ as

$$K_t + \varphi\varphi^\top = \left[V \;\; \frac{w}{||w||}\right] \Sigma' \left[V \;\; \frac{w}{||w||}\right]^\top \tag{4.9}$$

Algorithm 8 Approximate Stream Kernel *k*-means

1: **Input**:
- $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots\}, \mathbf{x}_i \in \Re^d$: the data stream to be clustered
- $\kappa(\cdot,\cdot) : \Re^d \times \Re^d \mapsto \Re$: the kernel function
- C: the initial number of clusters
- m: the initial number of points to be sampled ($m > C$)
- M: maximum number of points allowed in the sample set ($m < M$)
- γ: cluster decay rate
- η: cluster lifetime threshold

2: **Output**: Cluster labels for the data points in the stream
3: Initialize $S = \{\mathbf{x}_1\}$, $V_C = \mathbf{1}$ and $\Sigma_C = \kappa(\mathbf{x}_1, \mathbf{x}_1)$.
4: **for** $t = 1, 2, \ldots, m$ **do**
5: Set $S = S \cup \{\mathbf{x}_t\}$.
6: Update the eigenvalues Σ_C and eigenvectors V_C using (4.9).
7: **end for**
8: Cluster the data points in S into C clusters by executing *k*-means on ${V_C}{\Sigma_C}^{1/2}$.
9: Set the last update time $t_k = t, k \in [C]$.
10: Evaluate the recency function $f_k(t), k \in [C]$ according to (4.7).
11: **for** $t = m+1, m+2, \ldots$ **do**
12: Calculate the probability p_t using (4.2) and set $S = S \cup \{\mathbf{x}_t\}$ with probability p_t.
13: If $\mathbf{x}_t$ was added to S in Step 12, update the eigenvalues Σ_C and eigenvectors V_C using (4.9), and recluster the points in S by executing *k*-means on ${V_C}{\Sigma_C}^{1/2}$, otherwise find the cluster k^* whose center is closest to $\mathbf{x}_t$.
14: If $f_{k^*}(t) > \eta$, assign $\mathbf{x}_t$ to k^*, otherwise create a new cluster with $\mathbf{x}_t$ and set $C = C + 1$.
15: Find the clusters whose recency $f_k(t) \leq \eta, k \in [C]$, and remove these stale clusters. Set $C = C - c$, where c is the number of stale clusters.
16: 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.
17: **end for**

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 operation, repeated every time a new data point is input to the system, can be performed in $O(sC + C^3)$ time.

Clustering is performed every time a point is added to the sample set S, which takes $O(sC^2l)$ time, where l is the number of iterations required to reach convergence. In order to reduce the running time, we can employ a *lazy reclustering* approach, by which we perform the clustering after every T data point additions. To further enhance the efficiency of the algorithm, the data points can also be processed in batches of size B.

In summary, the time taken by the proposed approximate stream kernel *k*-means algorithm to cluster a data set of size n is $O\left(ndM + nCM + nC^3 + M^2C^2l\right) \sim O\left(nd + nC\right)$, when $\max(C, d, M, l) \ll n$. This contrasts with the $O(n^2)$ running time complexity of typical kernel-based clustering.

4.5 Experimental Results

4.5.1 Data sets

The proposed stream clustering algorithm inputs the data set in batches, and can handle potentially unbounded data sets, hence the size of the data set is not significant. The dimensionality of the data set plays an important role in the kernel similarity computation and the eigensystem update. We demonstrate the effectiveness of the proposed algorithm on the CIFAR-10, MNIST, Forest Cover Type, Imagenet-34, Poker, and Network Intrusion data sets.

4.5.2 Baselines

We compared the performance of the proposed algorithm with two recent stream clustering algorithms (StreamKM++ and sKKM), which have been shown to perform better than the other stream clustering algorithms. The StreamKM++ algorithm [6] is a linear stream clustering algorithm, which in the same spirit as the proposed algorithm, extracts the core points in the streaming data, and uses these core points to determine the cluster centers. The algorithm maintains a set

of buckets, each of size m. Data points are added to the first bucket until m points are received. They are then recursively merged with the points in the subsequent buckets to form a coreset of m points, using a coreset tree. The coresets are finally clustered using the *k*-means++ algorithm [12] to obtain the cluster centers. The performance of this algorithm depends on the coreset size m.

The streaming kernel *k*-means (sKKM) algorithm proposed in [84] processes the data in chunks of size m. The initial data chunk is clustered using kernel *k*-means. Weighted kernel *k*-means is used to cluster the subsequent data chunks. The cluster centers from the preceding data chunk are used to obtain the weights. We show that the proposed approximate stream kernel *k*-means is more effective than these algorithms. We also compare the performance of the proposed algorithm with (i) the batch *k*-means algorithm to show that our algorithm achieves higher accuracy, and (ii) the batch kernel *k*-means algorithm to evaluate the loss in the cluster quality. We could execute the kernel *k*-means algorithm only on the medium-sized CIFAR-10 and MNIST data sets due to its quadratic time complexity. For the remaining data sets, we executed kernel *k*-means on a $50,000$-sized randomly selected subset of the data, and assigned the remaining points to the closest cluster centers. This gives us an approximation of the time taken to execute kernel *k*-means on the full data set. We finally evaluate the performance of the proposed approximate stream kernel *k*-means algorithm when each data point is sampled with probability $1/2$, and show that importance sampling plays a significant role in reducing the memory requirements and enhancing the clustering accuracy.

4.5.3 Parameters

We used the universal RBF kernel for the proposed algorithm and the kernel-based baseline algorithms on all the data sets. We tuned the kernel width using grid search in the range $[0, 1]$ to obtain best performance. For the proposed approximate stream kernel *k*-means algorithm, we varied the initial sample size from $m = 1,000$ to $m = 5,000$ in multiples of $1,000$, and the maximum buffer size from $M = 5,000$ to $M = 20,000$ in multiples of $5,000$, to constrain the memory used to

4 GB. We employed the lazy reclustering approach with T set to 50, and processed the data in batches of size $B = 10,000$. We set the cluster decay factor $\gamma = 0.5$ as suggested in [86], and varied the lifetime threshold η as $\eta = \exp(-\gamma\tau)$, where $\tau = \{1, 2, \ldots, 5\}$. The coreset size and chunk size parameters for the StreamKM++ and sKKM algorithms were varied from $1,000$ to $5,000$. The initial number of clusters C was set equal to the true number of classes in the data set, for all the algorithms.

We obtained the code for the StreamKM++ algorithm from the authors[4], and implemented the other algorithms in MATLAB. We executed each algorithm 10 times on a 2.8 GHz processor with the memory constrained to 4 GB for the stream clustering algorithms, and to 40 GB for the batch clustering algorithms. We present the mean and variance of the time taken for clustering (in milliseconds) and the clustering quality, measured in terms of the Silhouette coefficient and NMI [104], over these 10 runs. Different permutations of the data set were input to the clustering algorithms in each run.

4.5.4 Results

4.5.4.1 Clustering efficiency and quality

Clustering time for our algorithm is computed as the average time taken to assign a label to each data point. For the baseline algorithms, we computed this time by dividing the total time taken to cluster the data set by the number of points in the data set. Figures 4.3, 4.4 and 4.5 compare the running time, silhouette coefficient and NMI values, respectively, of the proposed algorithm with the baseline algorithms, when the parameters $m = 5,000$, $M = 20,000$ and $\tau = 1$. As expected, the proposed algorithm was faster than the batch kernel *k*-means algorithms and its approximation (described in Section 4.5.2) on most of the data sets, but took longer than the *k*-means algorithm, because our algorithm has to compute the kernel similarity and its top eigenvectors unlike the *k*-

[4]The code for StreamKM++ is available at http://www.algorithm-engineering.de/software-projects?view=project&task=show&id=17

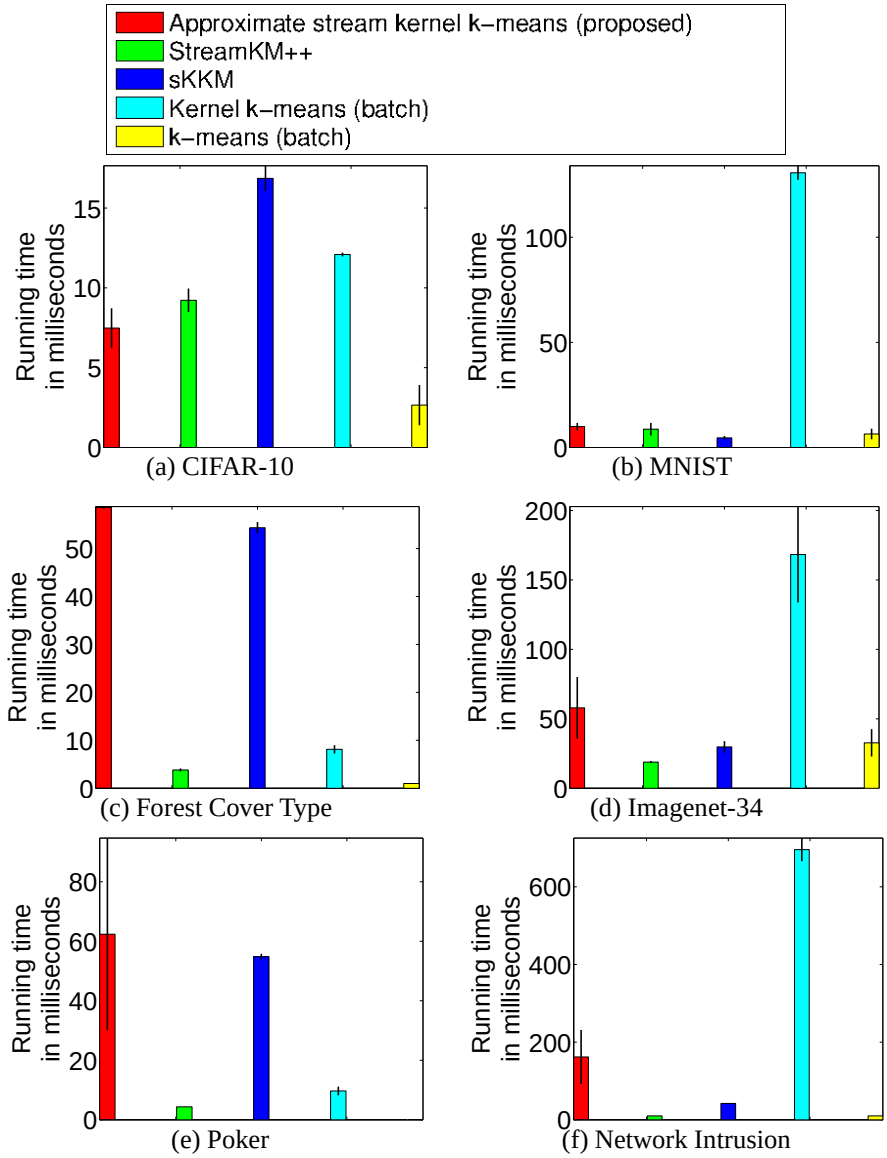


Figure 4.3 Running time (in milliseconds) of the stream clustering algorithms. The parameters for the proposed approximate stream kernel *k*-means algorithm are set to $m = 5,000$, $M = 20,000$, and $\tau = 1$. The coreset size for the StreamKM++ algorithm, and the chunk size of the sKKM algorithm are set to $5,000$. It is not feasible to execute kernel *k*-means on the Forest Cover Type, Imagenet-34, Poker, and Network Intrusion data sets due to their large size. The approximate running time of kernel *k*-means on these data sets is obtained by first executing kernel *k*-means on a randomly chosen subset of $50,000$ data points to find the cluster centers, and then assigning the remaining points to the closest cluster center.

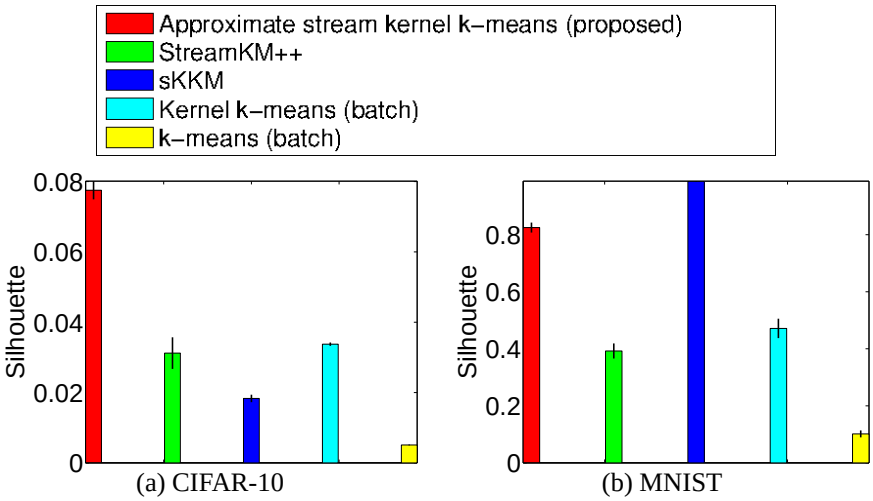


Figure 4.4 Silhouette coefficient values of the partitions obtained using the proposed approximate stream kernel *k*-means algorithm. The parameters for the proposed algorithm were set to $m = 5,000$, $M = 20,000$, and $\tau = 1$. The coreset size for the StreamKM++ algorithm, and the chunk size of the sKKM algorithm were set to $5,000$.

means algorithm. The silhouette coefficient values of the proposed algorithm are comparable to those of the kernel *k*-means, showing that they yielded similar partitions. The NMI achieved by our algorithm is higher than that of *k*-means because of the use of non-linear similarity measures. The proposed algorithm also outperforms the approximate variant of the kernel *k*-means algorithm, described in Section 4.5.2. On the CIFAR-10 data set, the batch kernel *k*-means achieved an NMI value of 16.9%. The proposed algorithm achieves comparable NMI values (15.5%).

Compared to the StreamKM++ algorithm, the proposed algorithm achieves higher clustering quality, both in terms of silhouette coefficient and NMI, although it takes slightly longer to assign cluster labels to the points. This is due to the fact that our algorithm needs to update and cluster the eigenvectors of the approximate kernel matrix for each batch of data points. The proposed algorithm offers the advantage that the cluster labels can be obtained in real-time, unlike the StreamKM++ algorithm which needs to process all the data points before assigning the cluster labels. For instance, the proposed algorithm was able to cluster about $2,700$ images from the CIFAR-10 data set per second, which is equivalent to a speed of about 8 MBps. On the remaining

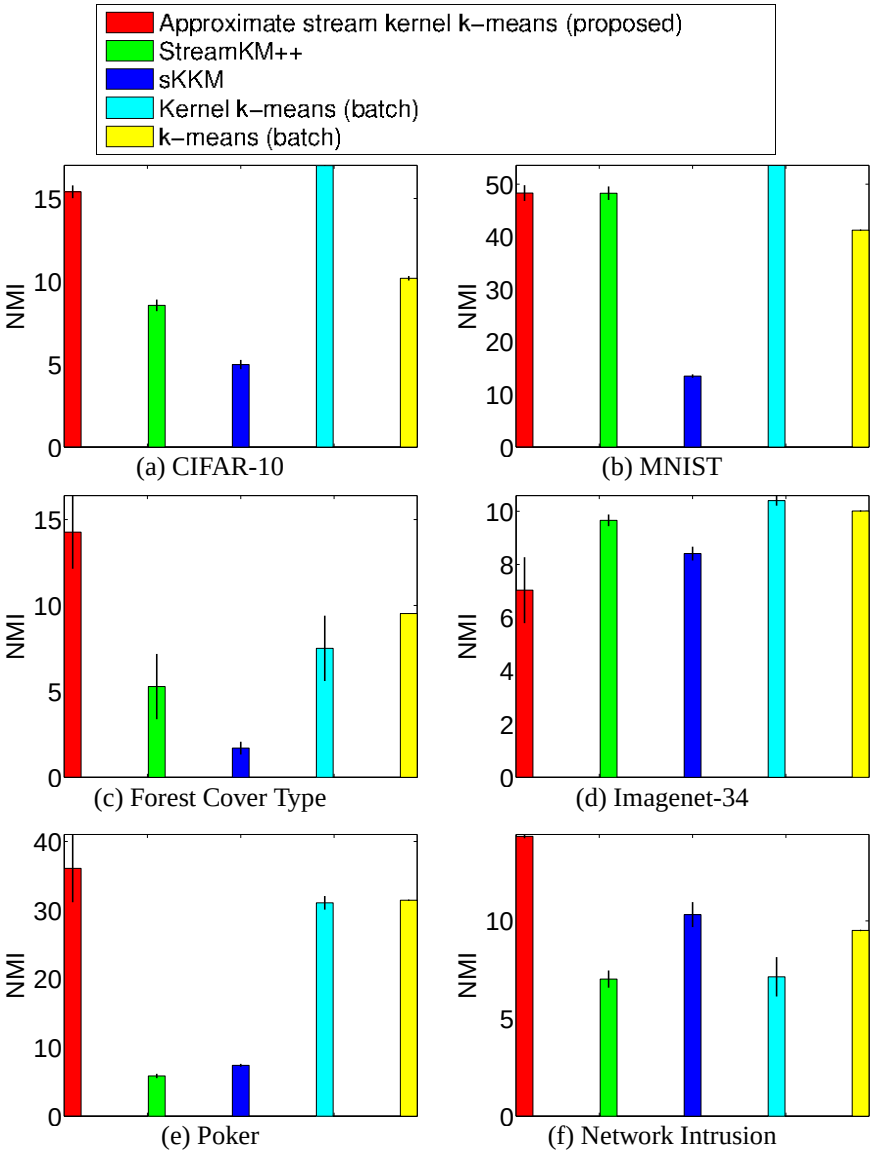


Figure 4.5 NMI (in %) of the clustering algorithms with respect to the true class labels. The parameters for the proposed approximate stream kernel *k*-means algorithm are set to $m = 5,000$, $M = 20,000$, and $\tau = 1$. The coreset size for the StreamKM++ algorithm, and the chunk size of the sKKM algorithm are set to $5,000$. It is not feasible to execute kernel *k*-means on the Forest Cover Type, Imagenet-34, Poker, and Network Intrusion data sets due to their large size. The approximate NMI values of kernel *k*-means on these data sets is obtained by first executing kernel *k*-means on a randomly chosen subset of $50,000$ data points to find the cluster centers, and then assigning the remaining points to the closest cluster center.

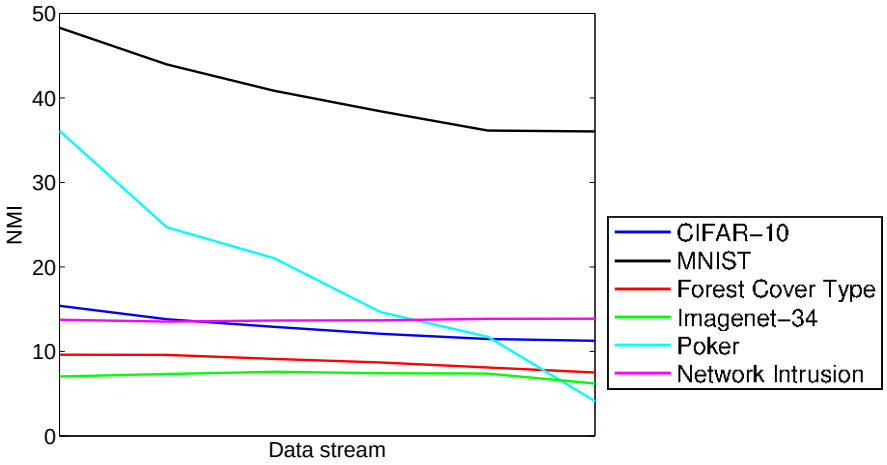


Figure 4.6 Change in the NMI (in %) of the proposed approximate stream kernel *k*-means algorithm over time. The parameters m, M and τ were set to $m = 5,000$, $M = 20,000$ and $\tau = 1$, respectively.

three data sets, the clustering speed ranges from 30 KBps to 700 KBps. Our algorithm also outperforms the sKKM clustering algorithm in terms of clustering quality. While the sKKM algorithm is slower than the proposed algorithm on the CIFAR-10 data set, it's speed is at par with the proposed algorithm on the remaining data sets. The StreamKM++ algorithm obtains clusters from coresets which summarize *all* the points in the data set. The sKKM algorithm relies on the information from only two time steps and discards most of the historical information. The proposed approximate stream kernel *k*-means algorithm finds the middle ground by retaining potentially useful data points using importance sampling, and discarding the rest of the data points. This is reflected in the silhouette and NMI values achieved by the algorithms.

Figure 4.6 shows how the NMI values of the proposed algorithm fall due to the accumulation of the kernel approximation error over time. We observe that the reduction in NMI is slow and stabilizes over time for most of the data sets, showing that the approximation error reduces over time. The error accumulation can be further minimized by clustering the points in the buffer more frequently (as discussed in Section 4.4), although this would increase the running time. The user can trade-off between the efficiency and accuracy by tuning the parameters of the algorithm.

4.5.4.2 Parameter sensitivity:

The proposed approximate stream kernel *k*-means algorithm relies on five parameters: initial sample size m, maximum buffer size M, initial number of clusters C, cluster decay rate γ and cluster lifetime threshold η. We study the influence of these parameters on the algorithm's performance and present heuristics to set the parameter values:

- **Initial sample size $\mathbf{m}$:** The time taken by the proposed algorithm to cluster each data point $\mathbf{x}_t$ is influenced by the number of points in the buffer S at time t, because the size of the eigenvector matrix V_C increases proportionally. The buffer size at time t, in turn, depends on the first m data points $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ input to the system. More data points are sampled from the stream and added to S, if the initial sample does not contain a sufficient number of representative points. On the CIFAR-10 data set, the number of additional points sampled reduced from $6,087$ to $4,434$ as the initial sample size m was increased from $1,000$ to $5,000$. Similar trends were observed for the remaining data sets as well. Figure 4.7 compares the running time of the proposed algorithm with the StreamKM++ and sKKM algorithms as the parameter m is varied. Recall that m represents the coreset size and the chunk size for the StreamKM++ and sKKM algorithms, respectively. As m was increased, the time taken for clustering by the baseline algorithms also increased. As expected, the proposed algorithm took slightly longer than the StreamKM++ and sKKM algorithms for most data sets, especially when m was large. However, the NMI values achieved by the proposed algorithm are much higher than those achieved by the baseline algorithms, as shown in Figure 4.9. Our algorithm's accuracy improves significantly as m increases, while there is minimal improvement in the cluster quality of the StreamKM++ algorithm. This improvement in accuracy compensates for the higher running time of the proposed algorithm. These results indicate that the initial sample, determined by the order of the data, plays a crucial role in the performance of the proposed algorithm. The variance in the NMI tends to reduce as m increases, again indicating that the order of the data is important. The silhouette coefficient values

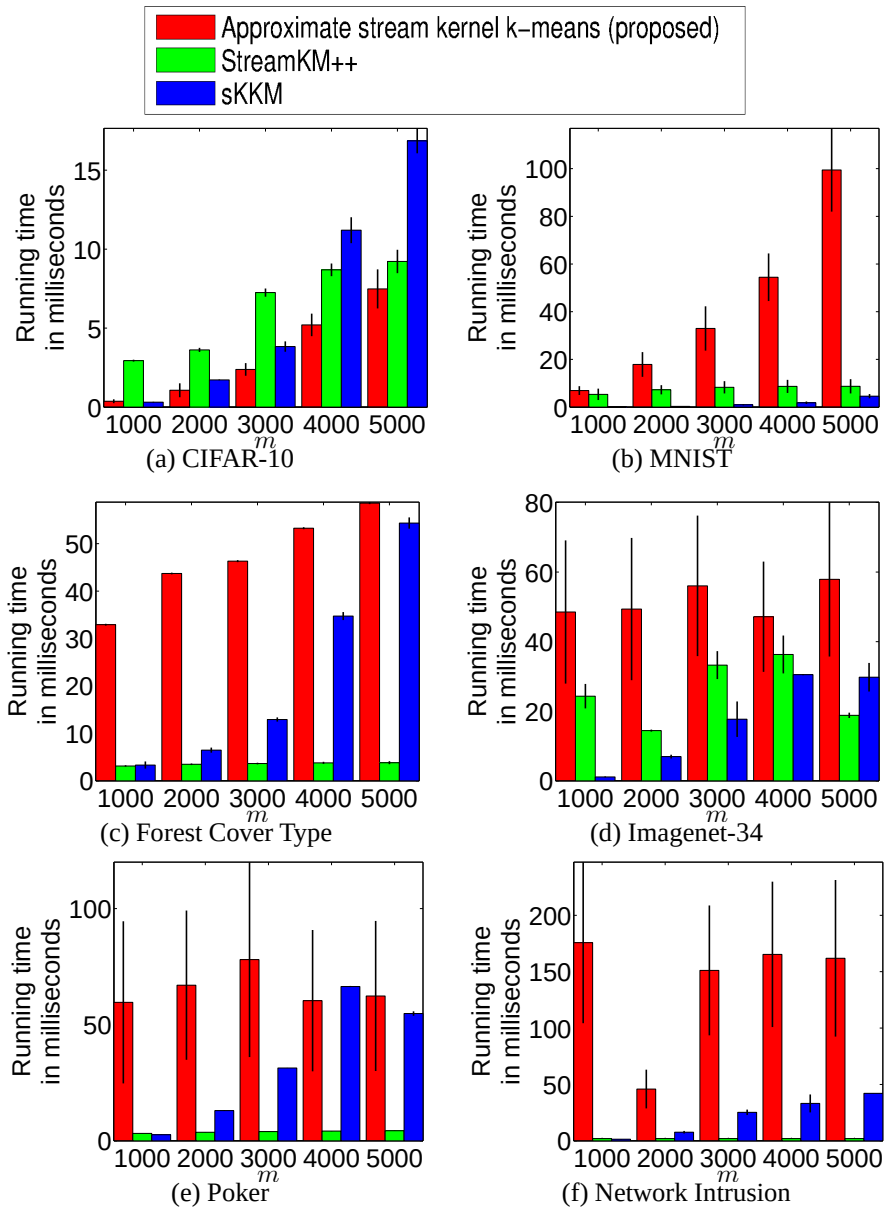
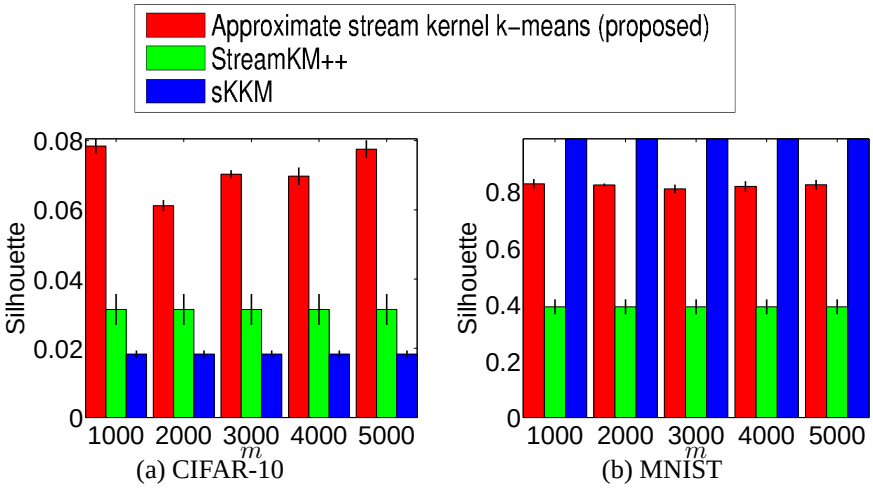


Figure 4.7 Effect of the initial sample size m on the running time (in milliseconds) of the proposed approximate stream kernel *k*-means algorithm. Parameter m represents the initial sample set size, the coreset size and the chunk size for the approximate stream kernel *k*-means, StreamKM++ and sKKM algorithms, respectively. The parameters M and τ are set to $M = 20,000$ and $\tau = 1$, respectively.



(a) CIFAR-10 (b) MNIST

Figure 4.8 Effect of the initial sample size m on the silhouette coefficient values of the proposed approximate stream kernel *k*-means algorithm. Parameter m represents the initial sample set size, the coreset size and the chunk size for the approximate stream kernel *k*-means, StreamKM++ and sKKM algorithms, respectively. The parameters M and τ are set to $M = 20,000$ and $\tau = 1$, respectively.

achieved by the proposed algorithm vary minimally with increase in the initial sample size, as shown in Figure 4.8.

- **Maximum buffer size M:** The maximum buffer size M does not affect the algorithmic efficiency of the proposed algorithm, provided that $M \sim 2m$, and the initial sample is representative of the stream. If M is small, data points need to be removed more often from the buffer to accommodate for the newly sampled data points, which results in an increased running time as shown in Table 4.2. For instance, when M was set to $5,000$, about $2,500$ points were removed from the buffer, whereas no points needed to be removed when $M = 20,000$, resulting in a 2 millisecond reduction of the clustering time per data point. The silhouette coefficient values vary minimally with M, as recorded in Table 4.3. The NMI value increases as M increases because a larger number of representative data points can be stored in the buffer, as shown in Table 4.4.
- **Cluster decay rate γ, lifetime threshold η and number of clusters C:** The final number of clusters at the end of clustering depends on the ordering of the data set, and the cluster

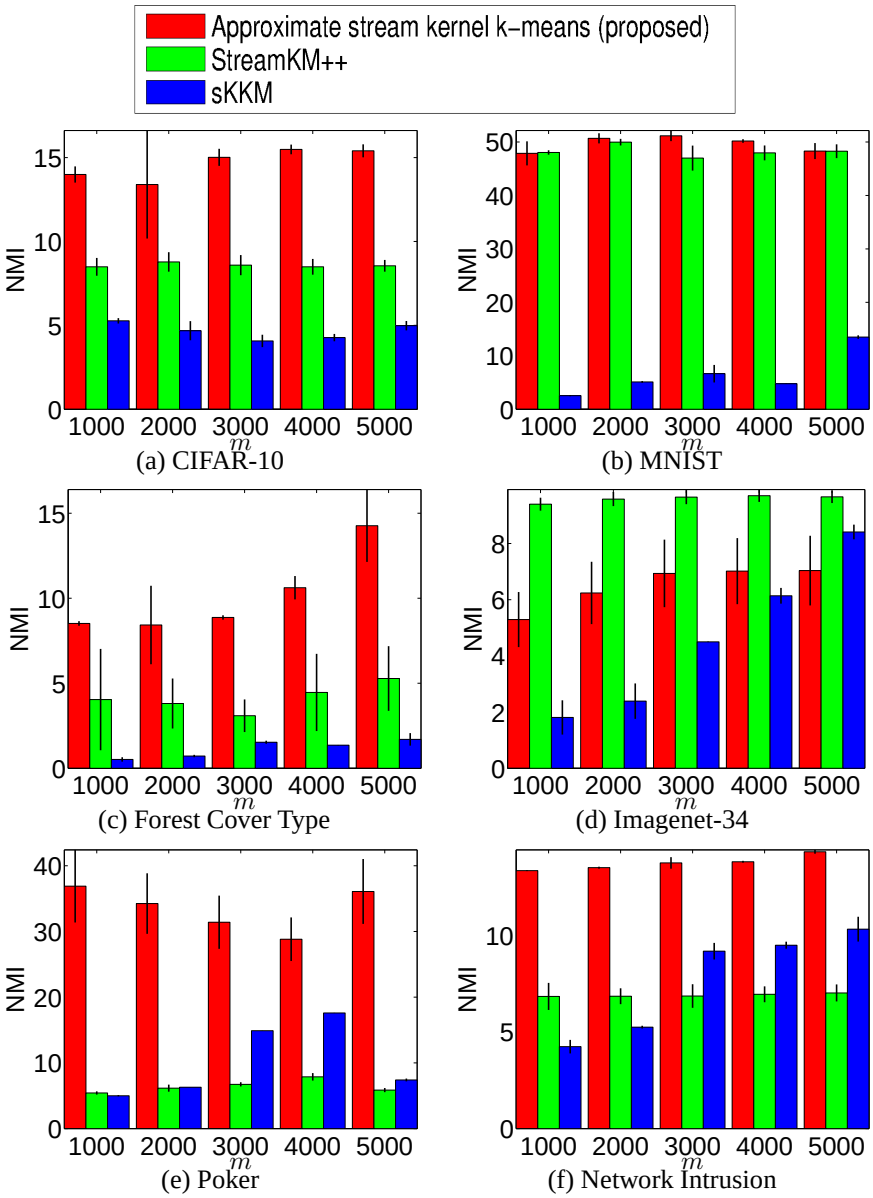


Figure 4.9 Effect of the initial sample size m on the NMI (in %) of the proposed approximate stream kernel *k*-means algorithm. Parameter m represents the initial sample set size, the coreset size and the chunk size for the approximate stream kernel *k*-means, StreamKM++ and sKKM algorithms, respectively. The parameters M and τ are set to $M = 20,000$ and $\tau = 1$, respectively.

Table 4.2 Effect of the maximum buffer size M on the running time (in milliseconds) of the proposed approximate stream kernel *k*-means algorithm. Parameter settings: $m = 5,000$, $\tau = 1$.

M	**5,000**	**10,000**	**15,000**	**20,000**
CIFAR-10	9.34 (± 0.76)	8.50 (± 3.33)	9.57 (± 2.79)	7.48 (± 1.24)
MNIST	11.05 (± 2.22)	10.35 (± 4.04)	8.99 (± 0.41)	9.94 (± 1.75)
Forest Cover Type	7.07 (± 0.27)	24.17 (± 6.69)	40.65 (± 12.81)	58.55 (± 21.57)
Imagenet-34	10.57 (± 2.62)	18.77 (± 4.85)	48.15 (± 18.18)	57.91 (± 22.20)
Poker	7.38 (± 3.56)	21.06 (± 9.57)	44.04 (± 17.76)	62.38 (± 32.31)
Network Intrusion	12.09 (± 2.57)	27.15 (± 7.07)	43.05 (± 15.31)	161.89 (± 69.43)

Table 4.3 Effect of the maximum buffer size M on the Silhouette coefficient of the proposed approximate stream kernel *k*-means algorithm. Parameter settings: $m = 5,000$, $\tau = 1$.

M	**5,000**	**10,000**	**15,000**	**20,000**
CIFAR-10 **($\times \mathbf{e-02}$)**	5.53 (± 0.12)	5.63 (± 0.04)	6.92 (± 0.29)	7.75 (± 0.26)
MNIST **($\times \mathbf{e-02}$)**	80.50 (± 0.84)	77.72 (± 0.66)	82.19 (± 1.29)	82.51 (± 1.75)

Table 4.4 Effect of the maximum buffer size M on the NMI (in %) of the proposed approximate stream kernel *k*-means algorithm. Parameter settings: $m = 5,000$, $\tau = 1$.

M	**5,000**	**10,000**	**15,000**	**20,000**
CIFAR-10	6.22 (± 0.27)	8.07 (± 2.73)	15.49 (± 0.18)	15.40 (± 0.39)
MNIST	20.15 (± 0.26)	29.97 (± 0.87)	48.31 (± 1.50)	48.31 (± 1.50)
Forest Cover Type	0.56 (± 0.07)	0.72 (± 0.05)	12.19 (± 0.02)	14.27 (± 2.13)
Imagenet-34	1.58 (± 1.27)	1.73 (± 1.62)	6.55 (± 1.19)	7.04 (± 1.24)
Poker	0.64 (± 3.45)	22.54 (± 2.92)	39.11 (± 4.19)	36.09 (± 4.94)
Network Intrusion	13.71 (± 0.01)	13.86 (± 0.40)	13.75 (± 0.30)	14.32 (± 0.10)

Table 4.5 Effect of the cluster lifetime threshold $\eta = \exp(-\gamma\tau)$ on the running time (in milliseconds) of the proposed approximate stream kernel *k*-means algorithm. Parameter settings: $m = 5,000$, $M = 20,000$.

τ	**1**	**2**	**3**	**4**	**5**
CIFAR-10	7.48 ($\pm$1.24)	9.28 ($\pm$1.03)	8.33 ($\pm$1.53)	8.54 ($\pm$1.66)	9.08 ($\pm$1.12)
MNIST	9.94 ($\pm$1.75)	9.25 ($\pm$0.46)	9.31 ($\pm$0.59)	9.42 ($\pm$0.61)	10.31 ($\pm$1.25)
Forest Cover Type	58.55 ($\pm$21.57)	42.80 ($\pm$17.26)	48.78 ($\pm$20.72)	40.09 ($\pm$13.81)	41.88 ($\pm$15.90)
Imagenet-34	57.91 ($\pm$22.20)	60.25 ($\pm$24.43)	55.77 ($\pm$26.20)	57.24 ($\pm$24.57)	54.98 ($\pm$31.10)
Poker	62.38 ($\pm$32.31)	44.39 ($\pm$16.04)	44.11 ($\pm$15.62)	42.65 ($\pm$17.48)	43.66 ($\pm$16.27)
Network Intrusion	161.89 ($\pm$0.69)	164.61 ($\pm$0.70)	165.18 ($\pm$0.71)	162.36 ($\pm$0.68)	163.05 ($\pm$0.64)

Table 4.6 Effect of the cluster lifetime threshold $\eta = \exp(-\gamma\tau)$ on the Silhouette coefficient of the proposed approximate stream kernel *k*-means algorithm. Parameters: $m = 5,000$, $M = 20,000$.

τ	**1**	**2**	**3**	**4**	**5**
CIFAR-10 **($\times$e $-$ 02)**	7.75 ($\pm$0.26)	7.66 ($\pm$0.24)	6.40 ($\pm$0.19)	6.35 ($\pm$0.20)	6.07 ($\pm$0.22)
MNIST **($\times$e $-$ 02)**	82.51 ($\pm$1.75)	82.51 ($\pm$0.01.75)	82.51 ($\pm$1.75)	82.51 ($\pm$1.75)	82.51 ($\pm$1.75)

Table 4.7 Effect of the cluster lifetime threshold $\eta = \exp(-\gamma\tau)$ on the NMI (in %) of the proposed approximate stream kernel *k*-means algorithm. Parameters: $m = 5,000$, $M = 20,000$.

τ	**1**	**2**	**3**	**4**	**5**
CIFAR-10	15.49 ($\pm$0.39)	15.55 ($\pm$0.23)	15.41 ($\pm$0.33)	15.45 ($\pm$0.23)	15.50 ($\pm$0.25)
MNIST	48.31 ($\pm$1.40)	47.77 ($\pm$1.49)	49.45 ($\pm$1.48)	45.98 ($\pm$1.40)	47.74 ($\pm$1.49)
Forest Cover Type	14.27 ($\pm$2.13)	12.10 ($\pm$0.03)	12.11 ($\pm$0.03)	12.10 ($\pm$0.03)	12.10 ($\pm$0.03)
Imagenet-34	7.04 ($\pm$1.24)	7.04 ($\pm$1.24)	6.95 ($\pm$1.14)	6.95 ($\pm$1.14)	7.76 ($\pm$1.54)
Poker	36.09 ($\pm$4.94)	32.07 ($\pm$4.41)	32.07 ($\pm$4.41)	36.09 ($\pm$4.94)	32.07 ($\pm$4.41)
Network Intrusion	14.32 ($\pm$0.10)	13.65 ($\pm$0.06)	13.65 ($\pm$0.06)	13.65 ($\pm$0.06)	13.66 ($\pm$0.06)

Table 4.8 Comparison of the performance of the approximate stream kernel *k*-means algorithm with importance sampling and Bernoulli sampling.

Data set		CIFAR -10	MNIST	Forest Cover Type	Imagenet -34	Poker	Network Intrusion
Importance sampling	**Running time (ms)**	7.48 (± 1.24)	9.94 (± 1.75)	58.55 (± 21.57)	57.91 (± 22.20)	62.38 (± 32.31)	161.89 (± 0.69)
	Silhouette coefficient $(\times \mathbf{e}-\mathbf{02})$	7.75 (± 0.26)	82.51 (± 1.75)	-	-	-	-
	NMI (%)	15.49 (± 0.39)	48.31 (± 1.40)	14.27 (± 2.13)	7.04 (± 1.24)	36.09 (± 4.94)	14.32 (± 0.10)
	Number of points sampled	5,434 $(\pm 2,093)$	6,136 (± 34)	16,561 $(\pm 3,710)$	14,735 $(\pm 1,790)$	6,265 (± 132)	14,886 $(\pm 2,627)$
Bernoulli sampling	**Running time (ms)**	2091.50 (± 47.34)	2210.77 (± 58.05)	1257.03 (± 39.33)	3002.45 (± 77.97)	86.43 (± 1.86)	923.16 (± 40.41)
	Silhouette coefficient $(\times \mathbf{e}-\mathbf{02})$	0.72 (± 0.01)	8.11 (± 0.13)	-	-	-	-
	NMI (%)	11.33 (± 4.9)	14.35 (± 0.05)	3.93 (± 0.7)	4.97 (± 0.19)	2.90 (± 0.02)	6.50 (± 0.15)
	Number of points sampled	31,483 (± 717)	23,000 (± 203)	407,220 $(\pm 5,807)$	389,177 $(\pm 11,325)$	50,000 (± 100)	1,711,101 $(\pm 44,866)$

decay and lifetime parameters γ and η. For instance, when the points in the CIFAR-10 data set were input in their true order (i.e. all images from class i are input before all images from class j ($i < j$)) for $C = 5$, $\gamma = 0.5$ and $\eta = \exp(-\gamma) = 0.61$, 10 clusters were found. On the other hand, when the data was permuted randomly and clustered, there was no increase in the number of clusters because no clusters became stale. The number of clusters increased more rapidly when γ and η were set to lower values because the clusters became stale faster. This also influences the clustering time minimally. The effect of the parameter η on the running time is recorded in Table 4.5. While the silhouette coefficient values remain almost constant when η changes, the NMI values are better for lower values of γ and η as shown in Tables 4.6 and 4.7.

Sampling techniques: Table 4.8 shows how the performance of the proposed algorithm on the six data sets changes, when importance sampling is replaced by a sampling procedure where each data point is sampled with probability $1/2$, and no limit is placed on the size of the sample set. We record, for each sampling procedure, the running time in milliseconds, the NMI values and the average number of points stored in memory after all the data points have been clustered. We also record the silhouette coefficient values for the CIFAR-10 and MNIST data sets. For importance sampling, we set the initial sample set size $m = 5,000$ and the recency threshold parameter τ =1. We observe that the number of points sampled with Bernoulli sampling is much higher than that with importance sampling. For instance, about $31,483$ points are sampled from the CIFAR-10 data set when Bernoulli sampling is employed, whereas only about $5,434$ points are sampled using importance sampling. In addition, the cluster quality of Bernoulli sampling is much lower than that of importance sampling. This is because the kernel approximation error is much higher when the data is sampled with equal probability. The running time is also higher when compared to the proposed algorithm with importance sampling due to the large number of sampled points.

4.6 Applications: Twitter Stream Clustering

Twitter[5] is a popular microblogging social network for sharing information over the web. It has over 100 million active users posting over $100,000$ short messages (called *tweets*) per minute, which include personal updates, real-time information about events, news etc. Each tweet contains a text message limited to 140 characters, and can include user-mentions, links, and emoticons in addition to plain text. Tweets are also often annotated with *hashtags* that denote keywords related to the tweets. A large body of work on topic detection, event detection, hashtag recommendation, and sentiment analysis has been performed on the Twitter data [16, 144]. Clustering has been used to find trending topics in Twitter posts, find user communities based on the similarity of their posts,

[5] `www.twitter.com`

and automatically annotate tweets with hashtags [16, 144]. In order to demonstrate the practical

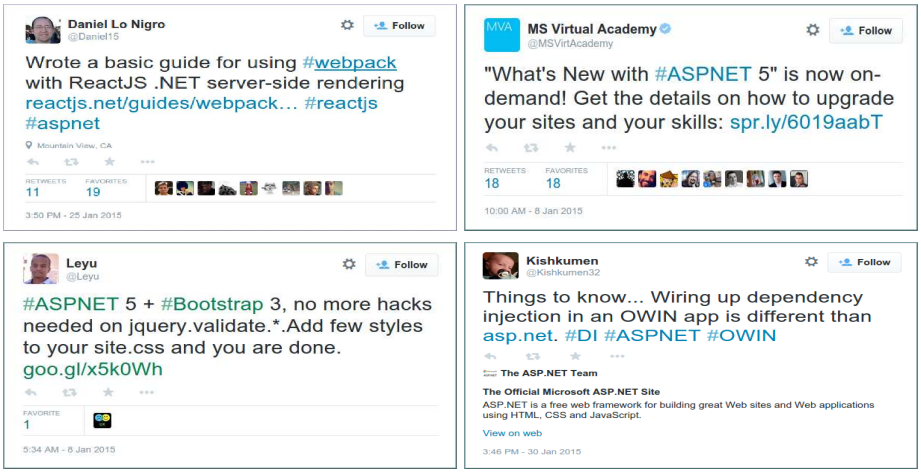


Figure 4.10 Sample tweets from the *ASP.NET* cluster.

applicability of the proposed approximate stream kernel *k*-means algorithm, we used it to cluster the Twitter data, and find the most-tweeted-about technologies over a period of time. We used the Twitter streaming search API to obtain over a billion tweets generated during the month of January 2015, using the following 20 popular keywords as hashtag search queries: Python, Perl, C#, Java, Ruby, C++, JavaScript, VBScript, Scala, Objective C, PHP, SQL, Postgresql, GO, Julia, Erlang, HTML, XML, Swift, and ASP.NET. We filtered out the non-English tweets, removed the hashtags and eliminated stop words to obtain a vocabulary containing $8,042$ terms. We used the corresponding tf-idf (term frequency-inverse document frequency) features [125], and the timestamp of the tweets as features for calculating the kernel, defined by

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

where ts_a and f_a denote the timestamp and the tf-idf features of a tweet, represented by data point $\mathbf{x}_a$, respectively. The first term in the kernel function ensures that two tweets which were generated in the same time period are likely to be assigned to the same cluster, and the second term ensures that two tweets with similar vocabulary are grouped together. We gave equal importance to both

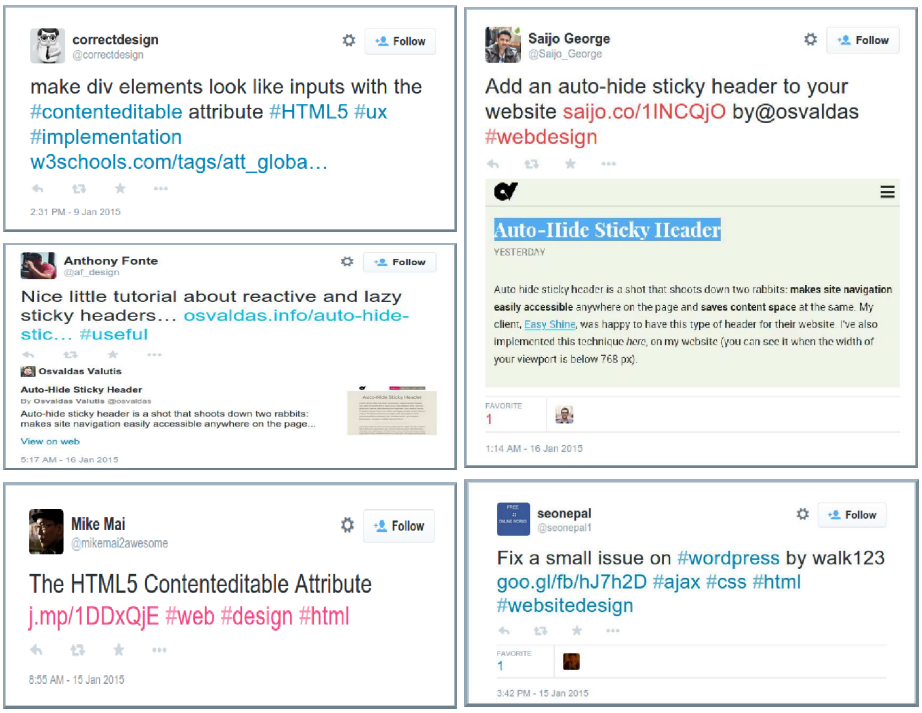
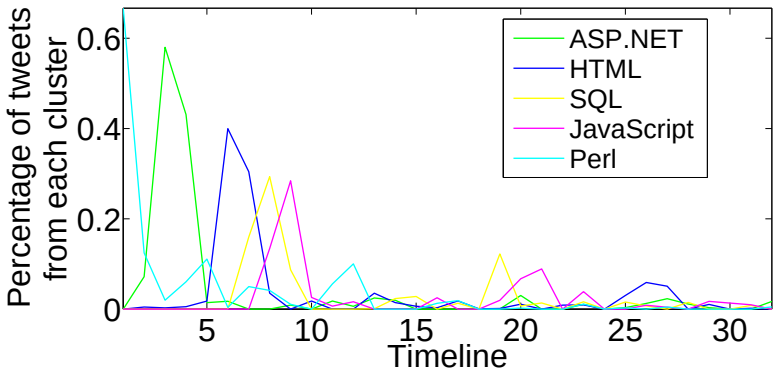


Figure 4.11 Sample tweets from the *HTML* cluster.

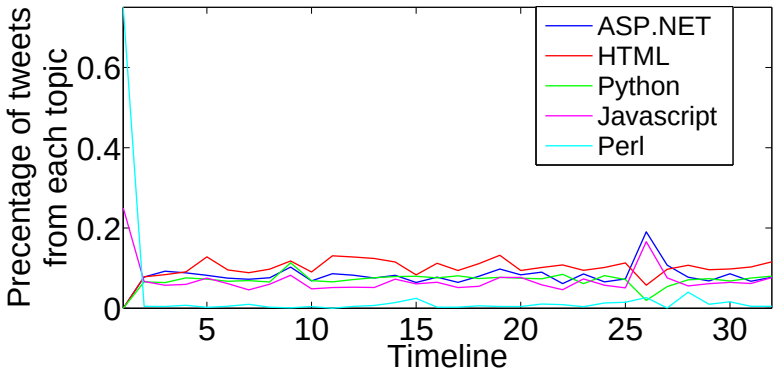
the timestamp and the tf-idf features by setting $\lambda = 0.5$. We set the parameters $m = 5,000$, $M = 10,000$, $C = 20$, $\gamma = 0.5$, $\eta = \exp(-\gamma) = 0.6$ and $B = 10,000$. Our algorithm assigned a cluster label to each tweet in about 200 milliseconds. Treating the hashtags as the ground truth labels[6], we obtained an average clustering accuracy of 61% in terms of NMI. On the other hand, the StreamKM++ algorithm took about 83 milliseconds per tweet, and achieved an NMI value of 40%, and the sKKM algorithm took about 2 seconds per tweet, and achieved an NMI value of 53%. Figures 4.10 and 4.11 show some sample tweets from the ASP.NET and HTML clusters, respectively. We observed that, by giving equal importance to the timestamp of the tweet, and the words in the tweet, we obtain clusters containing tweets that have both temporal proximity and vocabulary similarity. Retweets are always assigned to the same cluster as the original tweet. For example, the tweets about *sticky headers* are assigned to the HTML cluster, as seen in Figure 4.11.

[6]Although hashtags are prone to error, they are the best indicators of the topic of a tweet. They have been used as topic labels in many other studies including [75, 150].

More recent tweets rather than old tweets are stored in the memory. Figure 4.12(a) shows the trends of the top-five clusters over the month. This coincides well with the true trend of the top topics shown in Figure 4.12(b). We found that the order of popularity of the topic clusters is ASP.NET, HTML, SQL, JavaScript, Perl, C++, Postgresql, Python, GO, PHP, Swift, Scala, Java, Ruby, C#, XML, Erlang, Julia, Objective C and VBScript; while the true order of topic popularity is ASP.NET, HTML, Python, JavaScript, Perl, Java, PHP, Ruby, SQL, C++, Swift, C#, Scala, Postgresql, XML, Erlang, Julia, GO, Objective C, and VBScript.



(a) Cluster trends



(b) True trends of the topics

Figure 4.12 Trending clusters in Twitter. The horizontal axis represents the timeline in days and the vertical axis represents the percentage ratio of the number of tweets in the cluster to the total number of tweets obtained on the day. Figure (a) shows the trends obtained by the proposed approximate stream kernel *k*-means algorithm, and Figure (b) shows the true trends.

4.7 Summary

In this chapter, we have proposed an efficient and effective real-time kernel-based stream clustering algorithm, called approximate stream kernel *k*-means. Experimental results show that the proposed algorithm offers a good trade-off between clustering efficiency and clustering quality. Further, unlike some state-of-the-art kernel-based stream clustering algorithms, the proposed algorithm can control the decay and birth of clusters, thereby dynamically controlling the final number of clusters. The key to the efficiency of the proposed algorithm is the sampling of the streaming data based on their importance, defined in terms of the statistical leverage scores. This allows us to maintain the long-term history of the streaming data and also limit the memory required to store the data. We cater to the drift in the data distribution by placing thresholds on the life of a cluster. We demonstrated empirically that the proposed algorithm can cluster fast streams such as the Twitter stream with limited memory, and achieve higher clustering accuracy than the current stream clustering algorithms.

Chapter 5

Kernel-Based Clustering for Large Number of Clusters

5.1 Introduction

Document and image data sets, containing millions of 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 number of clusters. In this chapter, we present a variant of the online kernel clustering algorithm discussed in Chapter 4, called the *sparse kernel k-means algorithm* which can efficiently cluster large data sets into thousands of clusters, with significantly lower processing and memory requirements, and high clustering accuracy [38, 39].

Approximate kernel clustering algorithms such as approximate spectral clustering [67, 157] and approximate kernel *k*-means (from Chapter 2) 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 $O(nmd + nmC)$. Note that the running time increases proportionately with

Table 5.1 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 and Canopy algorithms are dependent on user-defined intra-cluster and inter-cluster distance thresholds, so their complexity is not directly dependent on C.

Clustering algorithms	**Complexity**
k-means [87]	$O(nCd)$
DBSCAN [61]	$O(n \log(n)\, d)$
Canopy [126]	$O(nCd)$
Kernel *k*-means [72]	$O(n^2 d + n^2 C)$
Spectral clustering [118]	$O(n^2 d + n^3 + nC^2)$
Support vector clustering [19]	$O(n^2 d n_{sv})$
Approximate spectral clustering [67]	$O(nmd + nmC)$
Approximate kernel *k*-means [40]	$O(nmd + nmC)$

the number of clusters (See Table 5.1). As demonstrated in Chapters 2 and 3, these algorithms take very long to cluster the data set when the number of clusters is in the order of thousands. In addition, the number of samples m required to obtain a good approximation is dependent on the rank of the kernel matrix, which is in turn dependent on the number of clusters in the data [74]. Clustering data sets with large number of clusters using these algorithms requires sampling $O(n)$ number of data points, to sufficiently represent all the clusters. This renders the approximate kernel clustering algorithms also non-scalable.

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 [34]. Fewer samples ($m = \Omega(C \log C)$) are required to construct a good kernel approximation using importance sampling than uniform sampling. 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 expensive [56]. 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 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 [132] 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 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.

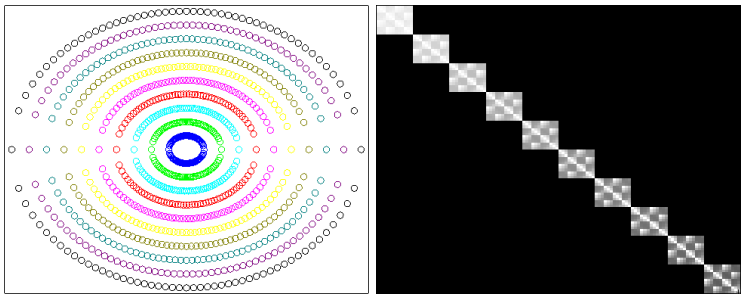
5.2 Background

Importance sampling As discussed in Chapter 4, the principle behind importance sampling is 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 $\Sigma_C = \text{diag}(\lambda_1, \ldots, \lambda_C)$ 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 sampled with probability p_i, defined as

$$p_i = \frac{1}{C} \left|\left| V_C^{(i)} \right|\right|_2^2, \tag{5.1}$$

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.

We showed in Lemma 8 that importance sampling reduces the dependency of the number of samples required on the number of clusters significantly, when compared to uniform sampling.



(a) (b)

Figure 5.1 Illustration of kernel sparsity on a two-dimensional synthetic data set containing $1,000$ points along 10 concentric circles. Figure (a) shows all the data points (represented by "o") and Figure (b) shows the RBF kernel matrix corresponding to this data. Neighboring points have the same cluster label when the kernel is defined correctly for the data set.

Kernel sparsity Another key component of the proposed algorithm is kernel sparsity. The proposed 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 5.1 illustrates this concept on the two-dimensional concentric circles data set. The RBF kernel matrix corresponding to this data is shown in Figure 5.1(b). Nearby data points in terms of the kernel similarity tend to have the same cluster label. This idea has been previously applied in several supervised local learning approaches [27]. The local learning-based clustering algorithm [188] and the local spectral clustering algorithm [118] also 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 s points would require the computation of $O(s)$ similarities. Popular approximate nearest neighbor algorithms adopt one of the following two approaches to find the nearest neighbors efficiently [3]:

- Use hashing techniques such as locality sensitive hashing, which use hash functions to place similar objects in the same bin [100, 198].

- Use data structures like *kd*-trees (also denoted as *k-d* trees) [131] and its variants like R-trees, R*-trees and metric trees [154] to organize the data according to their similarity and enable efficient querying.

The randomized *kd*-trees [132] 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 [131], 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 $O(s \log s)$ time to build the trees, and $O(\log s)$ time for each query. Therefore, the time taken for nearest neighbor computation is significantly reduced, when a large number of queries need to be performed on the same data set. We employ randomized *kd*-trees in the proposed algorithm to first find the nearest neighbors and build the sparse kernel matrix, and then 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 [40, 42, 67, 157, 188]:

(i) It employs importance sampling, so fewer number of samples are required to approximate the kernel matrix, when compared to the approximation methods in [40, 67, 157], which employ uniform random sampling.

(ii) Existing approximate kernel clustering algorithms [40, 67, 157] need to perform $O(nm)$ kernel similarity computations, where m is the number of samples. The number of kernel similarity computations performed by the proposed algorithm is $O(np)$, where the number of neighbors $p \ll m$. This also reduces the time and memory required for clustering, compared to the other approximate clustering algorithms.

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

(iv) It does not require the computation of the full kernel matrix, unlike the local clustering methods in [118] and [188].

(v) It 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).

5.3 Sparse Kernel k-means

The proposed sparse kernel *k*-means clustering algorithm is described in Algorithm 9. 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 p-neighbor graph K^0 for the m data points in S, defined by

$$\begin{aligned} K^0 &= [K_{ij}]_{m\times m} \text{ , where} \\ K_{i,j} &= \begin{cases} \kappa(\mathbf{x}_i, \mathbf{x}_j) & \text{if } \mathbf{x}_i \in \mathcal{N}(\mathbf{x}_j) \text{ and } \mathbf{x}_j \in \mathcal{N}(\mathbf{x}_i), \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{5.2}$$

We assume that nearby points in the Hilbert space belong to the same cluster. The kernel function should be appropriately defined for this assumption to be valid. Several articles in the literature describe techniques to learn the kernel function from the data [112, 177, 200].

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 $K^0 = V_C \Sigma_C {V_C}^\top$, where $\Sigma_C = \text{diag}(\lambda_1, \ldots, \lambda_C)$ contains the top C eigenvalues of K^0 and

[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 9 Sparse Kernel *k*-means

1: **Input**:
- $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}, \mathbf{x}_i \in \Re^d$: the set of n d-dimensional data points to be clustered
- $\kappa(\cdot,\cdot) : \Re^d \times \Re^d \mapsto \Re$: the 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

2: **Output**: Cluster labels for the data points
3: Initialize $S = \{\mathbf{x}_1 \ldots \mathbf{x}_m\}$.
4: 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 $\mathcal{D}^t = \{\mathbf{x}_1^t, \ldots, \mathbf{x}_B^t\}$.
5: Compute the sparse kernel matrix K^0 according to (5.2).
6: Decompose K^0 as $K^0 = V_C \Sigma_C V_C^\top$.
7: Cluster the data points in S by executing approximate *k*-means (Algorithm 10) on ${V_C \Sigma_C}^{1/2}$ to obtain their cluster labels.
8: **for** $t = 1, 2, \ldots, n_B$ **do**
9: **for** $i = 1, 2, \ldots, B$ **do**
10: Calculate the probability p_i^t using (5.1).
11: Set $S = S \cup \{\mathbf{x}_i^t\}$ with probability p_i^t.
12: If $\mathbf{x}_i^t$ was added to S in Step 11, update the eigenvalues Σ_C and eigenvectors V_C using (5.8), and recluster the points in S by executing the approximate *k*-means algorithm (Algorithm 10) on ${V_C \Sigma_C}^{1/2}$, otherwise assign $\mathbf{x}_i^t$ to cluster k^*, where $k^* = \arg\min_{k \in [C]} ||\mathbf{c}_k(\cdot) - g_t(\cdot)||^2_{\mathcal{H}_\kappa}$, $\mathbf{c}_k(\cdot)$ is given by (5.6), and $g_t(\cdot)$ is the projection of $\kappa(\mathbf{x}_l^i, \cdot)$ into the subspace spanned by the eigenvectors V_C.
13: 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.
14: **end for**
15: **end for**

$V_C = (\mathbf{v}_1, \ldots, \mathbf{v}_C)$ 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

$$K^t = \begin{cases} \begin{bmatrix} K^{t-1} & \varphi^\top \\ \varphi & \kappa(\mathbf{x}_i^t, \mathbf{x}_i^t) \end{bmatrix} & \text{with probability } p_i^t, \\ K^{t-1} & \text{with probability } 1 - p_i^t, \end{cases} \tag{5.3}$$

where φ is a sparse vector defined by $\varphi = [\kappa(\mathbf{x}_i^t, \mathbf{x}_s)]^\top$, $\mathbf{x}_s \in \mathcal{N}(\mathbf{x}_i^t) \cap S$, and p_i^t is the importance sampling probability defined in (5.1). 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

$$\max_{U \in \mathcal{P}} \text{tr}(\widetilde{U} K^t \widetilde{U}^\top), \tag{5.4}$$

where $U = (\mathbf{u}_1, \ldots, \mathbf{u}_C)^\top$ is the cluster membership matrix, $\widetilde{U} = [\text{diag}(U\mathbf{1})]^{-1/2} U$, domain $\mathcal{P} = \{U \in \{0,1\}^{C \times s} : U^\top \mathbf{1} = \mathbf{1}\}$, $s = \mathbf{card}(S)$, and $\mathbf{1}$ is a vector of all ones. 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 a smaller subspace, spanning the top eigenvectors of the kernel matrix K^t, along the lines of spectral clustering[2]. We pose the clustering problem as the following optimization problem:

$$\min_{U \in \mathcal{P}} \max_{\{\mathbf{c}_k(\cdot) \in \mathcal{H}_a\}_{k=1}^C} \sum_{k=1}^{C} \sum_{i=1}^{s} \frac{U_{k,i}}{s} ||\mathbf{c}_k(\cdot) - \kappa(\mathbf{x}_i, \cdot)||_{\mathcal{H}_\kappa}^2, \tag{5.5}$$

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

[2]Note that the eigenvalues and eigenvectors were computed while finding the sampling probabilities (5.1), hence the eigenvectors do not need to be re-computed for clustering.

eigenvectors of the kernel matrix:

$$\begin{aligned} \mathbf{c}_k(\cdot) &= \sum_{i=1}^{s}\sum_{j=1}^{C}\frac{U_{k,i}}{n_k}\sqrt{\lambda_j}\mathbf{v}_{i,j} \\ &= \frac{\mathbf{u}_k}{n_k}V_C\Sigma_C^{1/2},\ k \in [C], \end{aligned} \tag{5.6}$$

where n_k is the number of points in the k^{th} cluster, and $\mathbf{u}_k = (U_{k,1}, U_{k,2}, \ldots, U_{k,s})^\top$. By substituting (5.6) in (5.5), we obtain the following trace maximization problem:

$$\max_{U \in \mathcal{P}} \text{tr}(\widetilde{U}V_C\Sigma_C V_C^\top \widetilde{U}^\top). \tag{5.7}$$

The above problem can be solved by executing *k*-means on the matrix $V_C\Sigma_C^{1/2}$. The complexity of running *k*-means on $V_C\Sigma_C^{1/2}$ would be $O(sC^2)$, which can again be computationally expensive for large C.

We alleviate this issue by employing an approximate variant of the *k*-means algorithm (Algorithm 10), similar to the filtering algorithm in [91]. 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 cluster centers.

The proposed sparse kernel *k*-means 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 positive semi-definite and its rank is greater than the number of clusters C. Heuristics to set these parameters are discussed further in Section 5.5.

Algorithm 10 Approximate *k*-means

1: **Input**:
   - $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}, \mathbf{x}_i \in \Re^d$: the set of n d-dimensional data points to be clustered
   - C: the number of clusters
   - $MAXITER$: Maximum number of iterations
2: **Output**: Cluster labels for the data points
3: Randomly initialize the cluster labels $\{l_1, l_2, \ldots l_n\}, l_i \in [C]$.
4: Compute the cluster centers $\mathbf{c}_k = \sum_{l_i=k} \mathbf{x}_i, k \in [C]$.
5: Set $t = 0$;
6: **repeat**
7: Set $t = t + 1$.
8: Build randomized *kd*-tree index I for the C centers [132].
9: **for** $i = 1, 2, \ldots, n$ **do**
10: Find the approximate nearest center $\mathbf{c}_{k^*}$ of data point $\mathbf{x}_i$ using the index I.
11: Set $l_i = k^*$.
12: **end for**
13: Recompute the centers $\{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_C\}$.
14: **until** the labels do not change or $t > MAXITER$

5.4 Analysis

5.4.1 Computational 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 the time for computing the eigenvectors by making two modifications to the algorithm:

(i) Use efficient algorithms such as Lanczos, subspace iteration, and trace minimization methods to decompose the $m \times m$ kernel matrix K^0 obtained from the first m points [21]. 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 for the kernel matrix corresponding to the first m data points.

(ii) Update the eigenvectors V_C incrementally in each iteration of the algorithm using fast update mechanisms [31, 175], to reduce the time taken to process the points in each batch. Using the rank-1 update mechanism proposed in [31], we update the eigenvectors in $O(sp + p^3)$ time, where s is the number of points in the buffer S. Given the eigendecomposition, $K^t = V_C \Sigma_C V_C^\top$, and vector $\varphi \in \Re^m$, this method finds the eigendecomposition of $\left(K^t + \varphi\varphi^\top\right)$ as

$$K^t + \varphi\varphi^\top = \left[V \;\; \frac{w}{||w||}\right] \Sigma' \left[V \;\; \frac{w}{||w||}\right]^\top \tag{5.8}$$

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 (5.3) 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. Clustering is performed every time a point is added to the sample set S from the input batch of data points. In order to further reduce the running time, we can employ a *lazy reclustering* approach, by which we perform the clustering after every T 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 $O(npd + mp + m + QC \log Cl + QM \log Cl + n \log C)$, where Q is the total number of points sampled from the stream. We demonstrate in Section 5.5 that the number of points Q is close to the initial sample size m. Therefore, the running time complexity can be simplified as $O(npd + mp + m + mC \log Cl + mM \log Cl + n \log C) \sim O(npd + n \log C)$, assuming $\max(mp, mCl, mMl) \ll n$. Therefore, the proposed algorithm has running time complexity linear in n, linear in d, and logarithmic in C. It is significantly faster than the kernel *k*-means algorithm and the approximate kernel clustering algorithms, which have $O(n^2d + n^2C)$ 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.

5.4.2 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 $n \times Q$ matrix, where Q is the total number of points sampled from the data set using importance sampling. In this section, we first bound this approximation error (due to importance sampling and sparsification), and then bound the error incurred due to the approximation (5.5) for clustering.

Theorem 7. *Let K be the $n \times n$ kernel matrix and let $\bar{K}$ be the $n \times Q$ kernel matrix between the n points in the data set and the Q 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*

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

Assuming $\gamma = O(\sqrt{Q})$ *and* $\kappa(\cdot,\cdot) \leq 1$,

$$\max_{1 \leq i \leq C} |\mathbf{v}_i - \mathbf{z}_i|_2 \leq \frac{9\Delta}{2(\lambda_C - \lambda_{C+1})}, \tag{5.9}$$

with probability $1 - \delta$.

Proof. We will first establish a relationship between the singular vectors of the sparse kernel matrix that is constructed by the proposed algorithm and the $n \times n$ kernel matrix $K^{sp} = \left[K^{sp}_{i,j}\right]_{n \times n}$ defined as follows:

$$K^{sp}_{i,j} = \begin{cases} \kappa(\mathbf{x}_i, \mathbf{x}_j) & \text{if } \mathbf{x}_i \in \mathcal{N}(\mathbf{x}_j) \text{ and } \mathbf{x}_j \in \mathcal{N}(\mathbf{x}_i), \\ 0 & \text{otherwise,} \end{cases}$$

and then use the fact that $|K^{sp}|_F \leq |K|_F$ to obtain the required result. Let $\widehat{Z} = (\widehat{z}_1, \ldots, \widehat{z}_n)$ represent the eigenvectors of K^{sp}, $X = (\widehat{z}_1, \ldots, \widehat{z}_C)$, and $Y = (\widehat{z}_{C+1}, \ldots, \widehat{z}_n)$. Let L_n be a linear operator that maps any function $f(\cdot)$ to a function $L_n[f](\cdot) \in \mathcal{H}_\kappa$ defined by

$$L_n[f](\cdot) = \frac{1}{n} \sum_{i=1}^{n} \kappa(\mathbf{x}_i, \cdot) f(\mathbf{x}_i). \tag{5.10}$$

The eigenfunctions [187] of L_n, which form the basis of the space $\mathcal{H}_\kappa$ are given by

$$\widehat{\varphi}_i(\cdot) = \frac{1}{\sqrt{\lambda_i n}} \sum_{j=1}^{n} \widehat{z}_{i,j} \kappa(\mathbf{x}_j, \cdot). \tag{5.11}$$

Similar to L_n, let L_Q represent the linear operator based on the sampled examples, defined by

$$L_Q[f](\cdot) = \frac{1}{Q} \sum_{i=1}^{Q} \kappa(\mathbf{x}_i, \cdot) f(\mathbf{x}_i). \tag{5.12}$$

We first prove a simpler result that establishes a relationship between the subspaces X and V_C, in the following lemma:

Lemma 10. *Let $\delta \in (0,1)$ be the smallest probability such that $(\lambda_C - \lambda_{C+1}) > 3\Delta$, where Δ is defined as*

$$\begin{aligned}\Delta &= \frac{2\lambda_1}{Q}\ln\frac{2}{\delta} + \gamma|K^{sp}|_F\sqrt{\frac{2\ln(2/\delta)}{Qn}} \\ &\leq \frac{2\lambda_1}{Q}\ln\frac{2}{\delta} + \gamma|K|_F\sqrt{\frac{2\ln(2/\delta)}{Qn}}\end{aligned}$$

There exists a matrix $P \in \mathbb{R}^{(n-C)\times C}$ satisfying

$$\|P\|_F \leq \frac{2\Delta}{\lambda_C - \lambda_{C+1} - \Delta},$$

such that $V_C = (X + YP)(I + P^\top P)^{-1/2}$.

Proof. The proof is based on the following results (Lemmas 11 and 12) from [166] and [165], respectively:

Lemma 11. *Let $(\lambda_i, \mathbf{v}_i), i \in [n]$ be the eigenvalues and eigenvectors of a symmetric matrix $A \in \mathbb{R}^{n\times n}$ ranked in the descending order of eigenvalues. Set $X = (\mathbf{v}_1, \ldots, \mathbf{v}_C)$ and $Y = (\mathbf{v}_{C+1}, \ldots, \mathbf{v}_n)$. Given a symmetric perturbation matrix E, let*

$$(X,Y)^\top E(X,Y) = \begin{pmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \end{pmatrix}.$$

Let $\|\cdot\|$ represent a consistent family of norms and set

$$\gamma = \|E_{21}\|, \delta = \lambda_C - \lambda_{C+1} - \|E_{11}\| - \|E_{22}\|.$$

If $\delta > 0$ and $\frac{\gamma}{\delta} < \frac{1}{2}$, then there exists a unique matrix $P \in \mathbb{R}^{(n-C)\times C}$ satisfying $\|P\| < \frac{2\gamma}{\delta}$, such

that

$$X' = (X + YP)(I + P^\top P)^{-1/2}, Y' = (Y - XP^\top)(I + PP^\top)^{-1/2}$$

are the eigenvectors of $A + E$.

Lemma 12. *Let* $\mathcal{H}_\kappa$ *be a Hilbert space and* ξ *be a random variable on* (Z, ρ) *with values in* $\mathcal{H}_\kappa$. *Assume* $\|\xi\| \leq M < \infty$ *almost surely. Denote* $\sigma^2(\xi) = \mathbb{E}(\|\xi\|^2)$. *Let* $\{z_i\}_{i=1}^Q$ *be independent random drawers of* ρ. *For any* $0 < \delta < 1$, *with confidence* $1 - \delta$,

$$\left\| \frac{1}{Q} \sum_{i=1}^{Q} (\xi_i - \mathbb{E}[\xi_i]) \right\| \leq \frac{2M \ln(2/\delta)}{Q} + \sqrt{\frac{2\sigma^2(\xi) \ln(2/\delta)}{Q}}. \tag{5.13}$$

Let $\Delta_C = \lambda_C - \lambda_{C+1}$. Define

$$\begin{aligned} A &= [\langle \kappa(\mathbf{x}_i, \cdot), L_n \kappa(\mathbf{x}_j, \cdot) \rangle_{\mathcal{H}_\kappa}]_{n \times n}, \\ B &= [\langle \kappa(\mathbf{x}_i, \cdot), L_Q \kappa(\mathbf{x}_j, \cdot) \rangle_{\mathcal{H}_\kappa}]_{n \times n}, \end{aligned}$$

and $E = B - A$. We have

$$\gamma = \|X^\top E Y\|_F, \ \delta = \Delta_C - \|X^\top E X\|_F - \|Y E Y\|_F.$$

Using the relationship $\widehat{\varphi}_i = \sqrt{\frac{1}{\lambda_i n}} \sum_{k=1}^n \widehat{z}_{i,k} \kappa(\mathbf{x}_k, \cdot)$, $i = 1, \ldots, n$, we have

$$\begin{aligned} [X^\top E Y]_{i,j} &= \widehat{z}_i^\top E \widehat{z}_j \\ &= \sum_{a,b=1}^{n} \widehat{z}_{a,i} \widehat{z}_{b,j} \langle \kappa(\mathbf{x}_i, \cdot), (L_n - L_Q) \kappa(\mathbf{x}_j, \cdot) \rangle_{\mathcal{H}_\kappa} \\ &= \sqrt{\lambda_i \lambda_j} \langle \widehat{\varphi}_i, (L_n - L_Q) \widehat{\varphi}_j \rangle_{\mathcal{H}_\kappa} \\ &= \langle \widehat{\varphi}_i, L_n^{1/2} (L_n - L_Q) L_n^{1/2} \widehat{\varphi}_j \rangle_{\mathcal{H}_\kappa}. \end{aligned}$$

We have similar results for $X^\top EX$ and $Y^\top EY$. Thus, we obtain γ and δ as

$$\begin{aligned}
\gamma &= \sqrt{\sum_{i=1}^{C}\sum_{j=C+1}^{n}\langle\widehat{\varphi}_i,(L_n^2-L_n^{1/2}L_QL_n^{1/2})\widehat{\varphi}_j\rangle_{\mathcal{H}_\kappa}^2} \\
&\leq \|L_n^{1/2}(L_n-L_Q)L_N^{1/2}\|_F \\
\delta &= \Delta_C - \sqrt{\sum_{i,j=1}^{C}\langle\widehat{\varphi}_i,(L_n^2-L_n^{1/2}L_QL_n^{1/2})\widehat{\varphi}_i\rangle_{\mathcal{H}_\kappa}^2} \\
&\quad -\sqrt{\sum_{i,j=C+1}^{n}\langle\widehat{\varphi}_i,(L_n^2-L_n^{1/2}L_QL_n^{1/2})\widehat{\varphi}_i\rangle_{\mathcal{H}_\kappa}^2} \\
&\geq \Delta_C - \|L_n^{1/2}(L_n-L_Q)L_n^{1/2}\|_F.
\end{aligned}$$

We substitute these bounds for γ and δ into Lemma 11 to obtain

$$\|P\|_F \leq \frac{2\|L_n^2-L_n^{1/2}L_QL_n^{1/2}\|_F}{\lambda_C-\lambda_{C+1}-\|L_n^2-L_n^{1/2}L_QL_n^{1/2}\|_F}.$$

We now bound $\|L_n^2-L_n^{1/2}L_QL_n^{1/2}\|_F$ using Lemma 12. Let $\eta_i[f](\cdot)=\kappa(\mathbf{x}_i,\cdot)f(\mathbf{x}_i)$ and $\xi_i = L_N^{1/2}\eta_iL_N^{1/2}$. We define M and σ^2 as $M=\max_{1\leq i\leq n}\|\xi_i\|_F$, and $\sigma^2=\mathbb{E}_i\left[\|\xi_i\|_F^2\right]$.

We have $M\leq\|L_n\|_2\|\eta_i\|_F=\lambda_1$ and

$$\begin{aligned}
\sigma^2 &= \mathbb{E}\left[\sum_{k=1}^{n}\langle\widehat{\varphi}_k,\xi_i^2\widehat{\varphi}_k\rangle_{\mathcal{H}_\kappa}\right] \\
&= \mathbb{E}\left[\sum_{k=1}^{n}\langle\widehat{\varphi}_k,L_n^{1/2}\eta_iL_n\eta_iL_n^{1/2}\widehat{\varphi}_k\rangle_{\mathcal{H}_\kappa}\right] \\
&- \mathbb{E}\left[\langle\kappa(\mathbf{x}_i,\cdot),L_n\kappa(\mathbf{x}_i,\cdot)\rangle_{\mathcal{H}_\kappa}\sum_{k=1}^{n}\langle\widehat{\varphi}_k,L_n^{1/2}\eta_iL_n^{1/2}\widehat{\varphi}_k\rangle_{\mathcal{H}_\kappa}\right] \\
&\leq \frac{\gamma^2}{n}\mathbb{E}\left[\sum_{k=1}^{n}\langle\widehat{\varphi}_k,L_n^{1/2}\eta_iL_n^{1/2}\widehat{\varphi}_k\rangle_{\mathcal{H}_\kappa}\right] \\
&\leq \frac{\gamma^2}{n}\|L_n\|_F^2 \leq \frac{\gamma^2|K^{sp}|_F^2}{n} \leq \frac{\gamma^2|K|_F^2}{n}.
\end{aligned}$$

We complete the proof by substituting the bounds for M and σ^2 into Lemma 12. $\square$

Now we prove Theorem 7 using the result of Lemma 10. We have

$$\begin{aligned}
&\max_{1\leq i\leq C} |\mathbf{v}_i - \mathbf{z}_i|_2 = \|V_C - X\|_2 \\
\leq\ & \|YP(I+P^\top P)^{-1/2}\|_2 + \|(I-(I+P^\top P)^{-1/2})X\|_2 \\
\leq\ & \|Y\|_2\|P\|_2 + \|I-(I+P^\top P)^{-1/2}\|_2|X|_2 \\
\leq\ & \|P\|_F + 1 - \frac{1}{\sqrt{1+\|P\|_F^2}} \\
\leq\ & \|P\|_F + 1 - \sqrt{1-\|P\|_F^2} \leq \|P\|_F + \frac{\|P\|_F^2}{2} \leq \frac{3}{2}\|P\|_F.
\end{aligned}$$

We obtain the required result using the fact that

$$\|P\|_F \leq \frac{2\Delta}{\lambda_C - \lambda_{C+1} - \Delta} \leq \frac{3\Delta}{\lambda_C - \lambda_{C+1}}.$$

We complete the proof by using the fact $|K|_F \leq 1$ to obtain the relation $\Delta = O\left(\frac{1}{Q} + \frac{1}{\sqrt{n}}\right)$, when $\gamma = O(\sqrt{Q})$. $\square$

In the following lemma, we show that the error incurred due to the approximation (5.5) is well-bounded, provided that the tail of the eigenspectrum is fast decaying, which is true for most real data sets [45]:

Lemma 13. *Let E and E_a represent the optimal clustering errors in* (5.4) *and* (5.7)*, respectively. We have*

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

Proof. Let $\{\mathbf{c}_k^*(\cdot)\}_{k=1}^C$ and U^* be the optimal solution to (5.4). Let $c_k^a(\cdot)$ represent the projection of c_k^* into the subspace $\mathcal{H}_a$. For any $\kappa(\mathbf{x}_i,\cdot)$, let $g_i(\cdot)$ and $h_i(\cdot)$ be the projections of $\kappa(\mathbf{x}_i,\cdot)$ into the

subspace $\mathcal{H}_a$ and span$(\mathbf{v}_{C+1}, \ldots, \mathbf{v}_s)$, respectively. We have

$$\begin{aligned} E_a &= \min_{U \in \mathcal{P}} \max_{\mathbf{c}_k(\cdot) \in \mathcal{H}_a} \sum_{k=1}^{C} \sum_{i=1}^{s} \frac{U_{k,i}}{s} ||\mathbf{c}_k(\cdot) - \kappa(\mathbf{x}_i, \cdot)||^2_{\mathcal{H}_\kappa} \\ &\leq \sum_{k=1}^{C} \sum_{i=1}^{s} \frac{U^*_{k,i}}{s} ||\mathbf{c}^a_k(\cdot) - \kappa(\mathbf{x}_i, \cdot)||^2_{\mathcal{H}_\kappa} \\ &\leq \sum_{k=1}^{C} \sum_{i=1}^{s} \frac{U^*_{k,i}}{s} \left(||c^a_k(\cdot) - g_i(\cdot)||^2_{\mathcal{H}_\kappa} + ||h_i(\cdot)||^2_{\mathcal{H}_\kappa} \right) \\ &\leq E + \frac{1}{s} \sum_{k=1}^{C} \sum_{i=1}^{s} ||h_i(\cdot)||^2_{\mathcal{H}_\kappa} \leq E + \sum_{i=C+1}^{s} \lambda_i. \end{aligned}$$

□

5.5 Experimental Results

5.5.1 Data sets

We demonstrate the effectiveness of the proposed sparse kernel *k*-means algorithm using the CIFAR-100, Imagenet-164, Youtube and Tiny data sets.

5.5.2 Baselines and Parameters

We compared the performance of the proposed algorithm with the kernel *k*-means [72] algorithm on the CIFAR-100 data set. It is infeasible to execute the kernel *k*-means algorithm on the other three data sets. We also evaluated its performance against the *k*-means algorithm. We show that although the proposed algorithm has a higher running time than *k*-means, it yields better clustering accuracy. Finally, we compared our algorithm with the approximate kernel *k*-means algorithm from Chapter 2, where the data is sampled with uniform probability, and a low rank approximate kernel

is constructed using the sampled data. We show that importance sampling and kernel sparsity play a significant role in reducing the time and memory requirements.

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 CIFAR-100, Tiny and Imagenet-164 data sets. 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 best performance.

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-164 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. The memory used was constrained to 60 GB. We present the results (mean and variance) over the 10 runs. Different permutations of the data set were input to the clustering algorithms in each run. We used the randomized *kd*-trees implementation in the FLANN library [132] 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 randomized *kd*-tree parameters were set as follows: the number of dimensions n_d to 5, the number of trees to 8, and the approximation error to $\epsilon = 1e^{-16}$.

Table 5.2 Running time (in seconds) of the proposed sparse kernel *k*-means and the three baseline algorithms on the four data sets. The parameters of the proposed algorithm were set to $m = 20,000$, $M = 50,000$, and $p = 1,000$. The sample size m for the approximate kernel *k*-means algorithm was set equal to $20,000$ for the CIFAR-100 data set and $10,000$ for the remaining data sets. It is not feasible to execute kernel *k*-means on the Imagenet-164, Youtube and Tiny data sets due to their large size. The approximate running time of kernel *k*-means on these data sets is obtained by first executing the algorithm on a randomly chosen subset of $50,000$ data points to find the cluster centers, and then assigning the remaining points to the closest cluster center.

Dataset	**Sparse Kernel *k*-means (proposed)**	**Approx. kernel *k*-means**	***k*-means**	**Kernel *k*-means**
CIFAR-100	49,887 (± 93)	11,394 (± 600)	1,507 (± 332)	117,513 (± 211)
Imagenet-164	74,794 (± 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 $(\pm 8,482)$

5.5.3 Results

5.5.3.1 Running time

Table 5.2 compares the running time of our algorithm with the approximate kernel *k*-means, kernel *k*-means and *k*-means algorithms, when the parameters m and p are equal to $20,000$ and $1,000$, respectively. On the CIFAR-100 data set, the proposed algorithm takes longer than the *k*-means algorithm, as expected, because of the additional time required for kernel computation and eigensystem calculation. It also takes 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, and computes the cluster centers using the low-rank matrix constructed from this subset. The proposed algorithm, the approximate kernel *k*-means, and the *k*-means algorithms are significantly faster than the kernel *k*-means algorithm. The proposed algorithm spends more

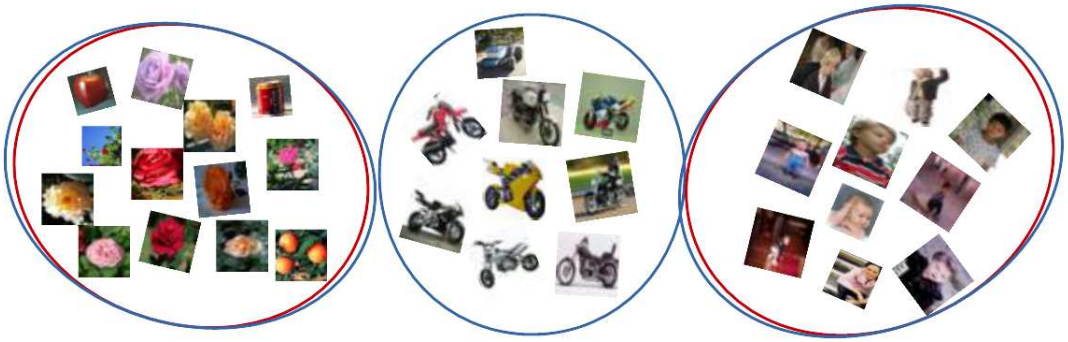


Figure 5.2 Sample images from three of the 100 clusters in the CIFAR-100 data set obtained using the proposed algorithm.

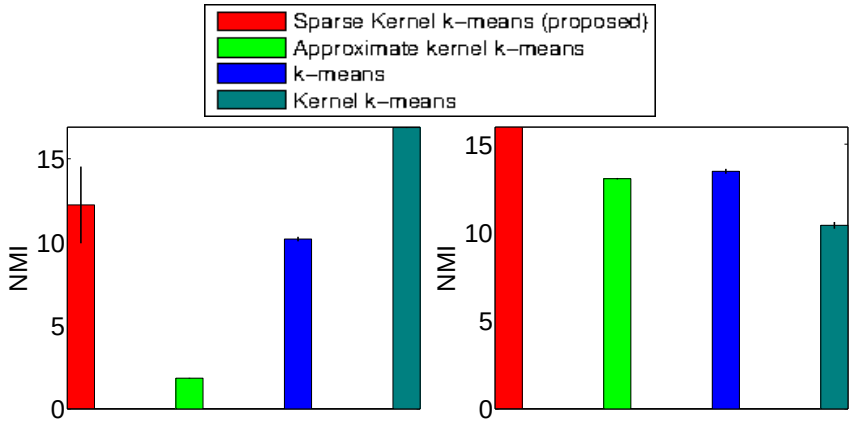
time in updating the eigenvectors and finding the leverage scores than clustering the eigenvectors to obtain the cluster labels. Similar performance is observed on the Imagenet-164, Youtube and Tiny data sets. The proposed algorithm is also faster than *k*-means on the Imagenet-164 data set, because *k*-means takes longer to converge. It is infeasible to compute the full kernel matrix for the Imagenet-164, Youtube and Tiny data sets, so we were unable to execute kernel *k*-means on them. For these data sets, we executed kernel *k*-means on a $50,000$-sized randomly selected subset of the data, and assigned the remaining points to the closest cluster centers. The proposed algorithm is also faster than this implementation of kernel *k*-means, because it takes a long time to find the distance between the data points and the cluster centers, and assign labels. The proposed algorithm is also more accurate than this kernel *k*-means implementation on the Imagenet-164 data set.

5.5.3.2 Cluster quality

Figure 5.2 show examples of clusters obtained, using the sparse kernel *k*-means algorithm, from the CIFAR-100 data set. We assigned a class label to each cluster, based on the true class of majority of the objects in the cluster. Table 5.3 records the silhouette coefficient values of the partitions of the CIFAR-100 data set. The sparse kernel *k*-means algorithm achieves values closer to that of the kernel *k*-means algorithm. The approximate kernel *k*-means and *k*-means algorithms are unable to

Table 5.3 Silhouette coefficient ($\times e-02$) of the proposed sparse kernel *k*-means and the three baseline algorithms on the CIFAR-100 data set. The parameters of the proposed algorithm were set to $m = 20,000$, $M = 50,000$, and $p = 1,000$. The sample size m for the approximate kernel *k*-means algorithm was set equal to $m = 20,000$.

Sparse kernel *k*-means (proposed)	**Approx. kernel *k*-means**	***k*-means**	**Kernel *k*-means**
11.36 (± 0.07)	2.33 (± 0.02)	3.02 (± 0.01)	30.18 (± 0.13)



(a) CIFAR-100 (b) Imagenet-164

Figure 5.3 NMI (in %) of the proposed sparse kernel *k*-means and the three baseline algorithms on the CIFAR-100 and Imagenet-164 data sets. The parameters of the proposed algorithm were set to $m = 20,000$, $M = 50,000$, and $p = 1,000$. The sample size m for the approximate kernel *k*-means algorithm was set equal to $20,000$ for the CIFAR-100 data set and $10,000$ for the Imagenet-164 data set. It is not feasible to execute kernel *k*-means on the Imagenet-164 data set, due to its large size. The approximate NMI value achieved by kernel *k*-means on the Imagenet-164 data set is obtained by first executing the algorithm on a randomly chosen subset of $50,000$ data points to find the cluster centers, and then assigning the remaining points to the closest cluster center.

achieve similar silhouette values.

We analyze the prediction accuracy, in terms of NMI, of the proposed sparse kernel *k*-means using the CIFAR-100 and Imagenet-164 data sets. As the true class labels for the Youtube and Tiny data sets are not available, we were unable to find the NMI for these data sets. Figure 5.3 shows the NMI values with respect to the true class labels, for each of the algorithms on the CIFAR-100 and Imagenet-164 data sets. In Figure 5.3(a), it is observed that the NMI achieved by our algorithm is close to that of the kernel *k*-means algorithm. The proposed algorithm outperforms both *k*-means and approximate kernel *k*-means on both the CIFAR-100 and Imagenet-164 data sets, due to the fact that it samples the most informative points from the data set.

5.5.3.3 Parameter sensitivity

Our sparse kernel *k*-means algorithm relies on three parameters: the initial sample set m, the maximum size of the sample set M, and the size of the neighborhood p. We evaluated the effect of each of these parameters on the performance of the proposed algorithm, using the CIFAR-100 and Imagenet-164 data sets.

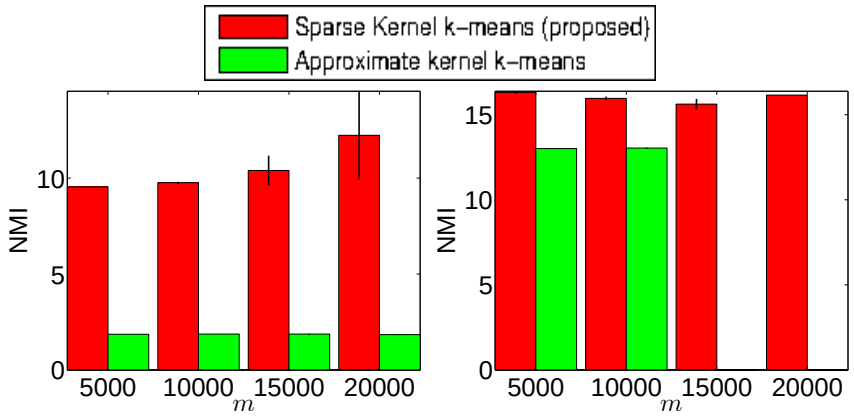
- **Initial sample:** The initial sample used to construct the kernel K^0, and obtain the initial cluster labels plays a crucial role in the performance of our algorithm as shown in Table 5.4, Table 5.5 and Figure 5.4. They compare the performance of the proposed algorithm and the approximate kernel *k*-means algorithm with increasing m value. As expected, the running time of both the algorithms increases as the initial sample size increases from $m = 5,000$ to $m = 20,000$. As m increases, the size of the initial kernel K^0, and the time to compute and decompose it into its eigenvalues and eigenvectors increase proportionately. The initial sample also determines the number of points sampled from the data set, 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. The time to cluster increases as more points are added to the buffer. The silhouette coefficient values on the CIFAR-100

Table 5.4 Comparison of the running time (in seconds) of the proposed sparse kernel *k*-means algorithm and the approximate kernel *k*-means algorithm on the CIFAR-100 and the Imagenet-164 data sets. Parameter m represents the initial sample set size for the proposed algorithm, and the size of the sampled subset for the approximate kernel *k*-means algorithm. The remaining parameters of the proposed algorithm are set to $M = 50,000$, and $p = 1,000$. Approximate kernel *k*-means is infeasible for the Imagenet-164 data set when $m > 10,000$ due to its large size.

m	**CIFAR-100**		**Imagenet-164**	
	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 $(\pm 4,469)$	15,691 $(\pm 3,786)$
10,000	18,256 (± 21)	4,134 (± 549)	36,669 (± 603)	16,023 $(\pm 3,577)$
15,000	34,192 $(\pm 2,652)$	7,856 (± 929)	53,142 $(\pm 3,058)$	-
20,000	49,887 (± 93)	11,394 (± 600)	74,794 (± 870)	-

Table 5.5 Comparison of the silhouette coefficient ($\times e-02$) of the proposed sparse kernel *k*-means algorithm and the approximate kernel *k*-means algorithm on the CIFAR-100 data set. Parameter m represents the initial sample set size for the proposed algorithm, and the size of the sampled subset for the approximate kernel *k*-means algorithm. The remaining parameters of the proposed algorithm were set to $M = 50,000$, and $p = 1,000$.

m	5,000	10,000	15,000	20,000
Sparse kernel *k*-means (proposed)	19.42 (± 0.12)	11.77 (± 0.04)	11.67 (± 0.06)	11.36 (± 0.07)
Approx. kernel *k*-means	2.45 (± 0.03)	2.37 (± 0.02)	2.45 (± 0.02)	2.33 (± 0.02)



(a) CIFAR-100 (b) Imagenet-164

Figure 5.4 Comparison of the NMI (in %) of the proposed sparse kernel *k*-means algorithm and the approximate kernel *k*-means algorithm on the CIFAR-100 and the Imagenet-164 data sets. Parameter m represents the initial sample set size for the proposed algorithm, and the size of the sampled subset for the approximate kernel *k*-means algorithm. The remaining parameters of the proposed algorithm were set to $M = 50,000$, and $p = 1,000$. Approximate kernel *k*-means is infeasible for the Imagenet-164 data set when $m > 10,000$ due to its large size.

data set decrease minimally when m increases from $5,000$ to $10,000$, but remain constant for $m \geq 10,000$. On the other hand, there is minimal change in the silhouette values of the approximate kernel *k*-means algorithm for increasing m. The NMI values achieved by our algorithm increase considerably as the sample size m increases, indicating that the initial sample is important to the clustering accuracy of the proposed algorithm. Even with just $5,000$ data points in the initial sample, our algorithm is able to achieve 13% NMI. On the other hand, the approximate kernel *k*-means algorithm is unable to achieve the same with even $20,000$ samples. The performance of the sparse kernel *k*-means algorithm is best when the sample size is set greater than $C \log C$, in accordance with Lemma 8.

- **Maximum sample size:** In our experiments, we set the maximum sample size to $50,000$. We found that this parameter is not as critical as the initial sample, provided that it is set large enough to accommodate for a sufficiently representative sample. On both the CIFAR-100 and Imagenet-164 data sets, the number of points added to the buffer range from 100 to 500, on an average. The number of points added decreases as the initial sample size m increases

Table 5.6 Effect of the size of the neighborhood p on the running time (in seconds), the silhouette coefficient and NMI (in %) of the proposed sparse kernel *k*-means algorithm on the CIFAR-100 and Imagenet-164 data sets. The remaining parameters of the proposed algorithm were set to $m = 20,000$, and $M = 50,000$.

p	CIFAR-100			Imagenet-164	
	Running time	**Silhouette coefficient** ($\times e-02$)	**NMI**	**Running time**	**NMI**
1,000	49,887 (± 93)	11.36 (± 0.07)	12.23 (± 2.3)	74,794 (± 870)	16.15 (± 0.004)
5,000	52,073 (± 483)	11.25 (± 0.06)	12.09 (± 0.02)	82,880 ($\pm 21,360$)	17.58 (± 0.10)
10,000	54,205 (± 874)	12.27 (± 0.12)	13.86 (± 0.07)	192,725 ($\pm 3,874$)	18.01 (± 0.07)
15,000	55,062 (± 837)	11.32 (± 0.09)	14.00 (± 0.01)	247,911 ($\pm 7,789$)	18.23 (± 0.004)

from $5,000$ to $20,000$. For instance, on the CIFAR-100 data set, when $m = 5,000$, 453 additional points were added to the buffer. When $m = 20,000$, only 69 points were added.

- **Size of the neighborhood:** The number of neighbors p used to construct the sparse kernel similarity is also important to the performance of the proposed sparse kernel *k*-means algorithm. Table 5.6 shows how the running time, the silhouette coefficient values, and the NMI values on the CIFAR-100 and Imagenet-164 data sets are affected as the value of p increased from $1,000$ to $15,000$, and the initial sample size m was fixed at $20,000$. The running time doubles when p increased from $p = 1,000$ to $p = 15,000$, on both the data sets. This is due to the fact that a larger number of similarity computations need to be performed as the value of p increases. However, although there is a small increase in the silhouette coefficient and NMI values, the increase is not significant enough to justify the increase in the running time. We conclude that the neighborhood size is an important parameter in determining the efficiency of the algorithm.
- **Number of clusters:** We show the effect of varying the number of clusters C on the performance of the proposed algorithm, in Figures 5.5 and 5.6. The running time of the algorithm

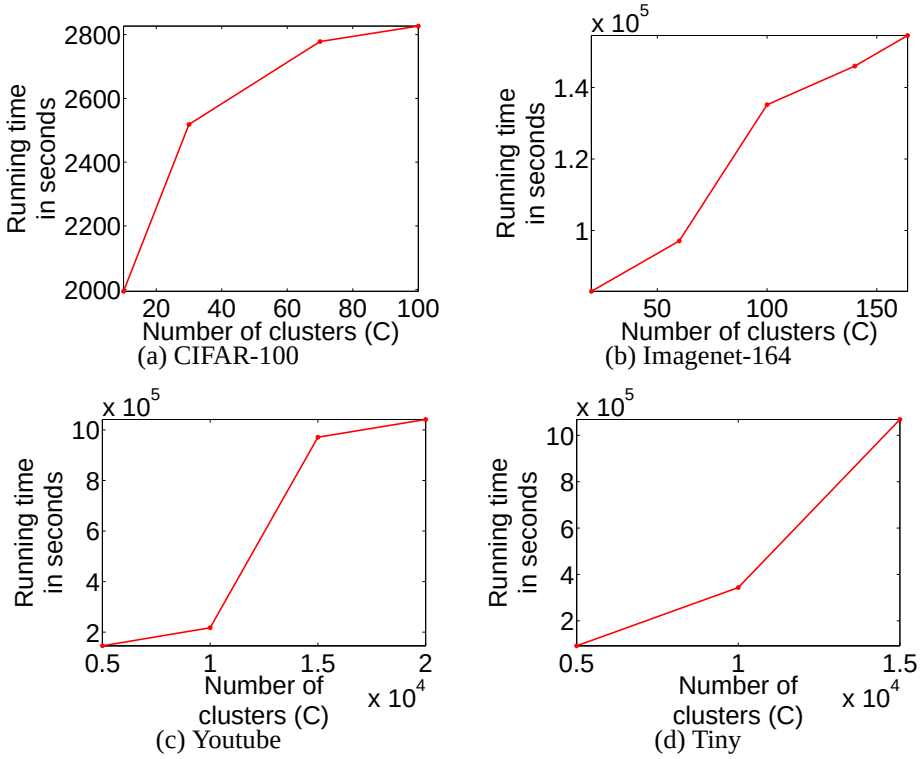


Figure 5.5 Effect of the number of clusters C on the running time (in seconds) of the proposed sparse kernel *k*-means algorithm.

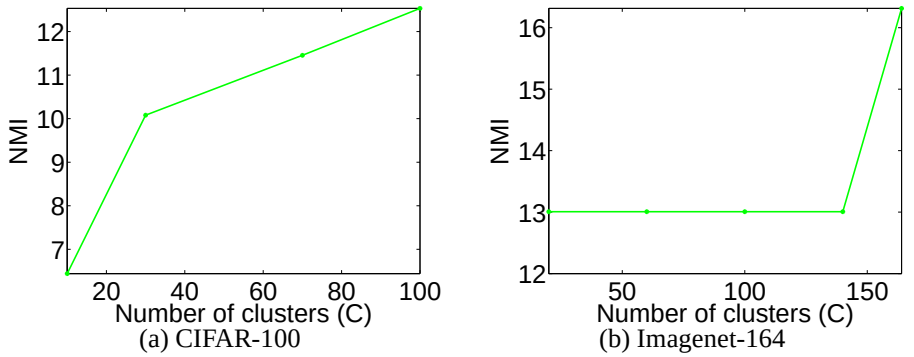


Figure 5.6 Effect of the number of clusters C on the NMI (in %) of the proposed sparse kernel *k*-means algorithm.

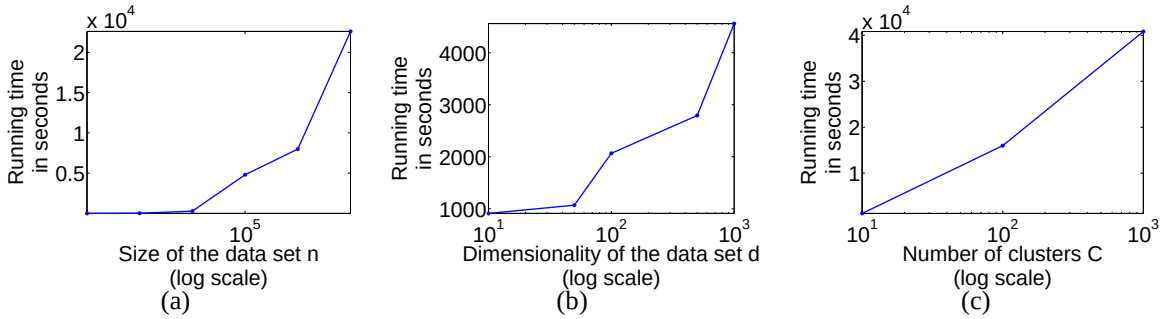


Figure 5.7 Running time of the sparse kernel *k*-means clustering algorithm for different values of (a) n, (b) d and (c) C.

increases with the number of clusters. However, unlike many other clustering algorithms including the approximate kernel *k*-means, RFF, SV and approximate stream kernel *k*-means clustering algorithms presented in the earlier chapters, the running time of the sparse kernel *k*-means algorithm increases almost logarithmically with the number of clusters, on most data sets. The NMI values achieved by our algorithm also increase as the number of clusters increase. We note that the NMI values of the proposed algorithm are better than those achieved by the baseline algorithms, on both the CIFAR-100 and Imagenet-164 data sets, for all values of C.

5.5.3.4 Scalability

We varied the number of points, the dimensionality and the number of clusters in the concentric circles data set, and executed our algorithm on these data sets to examine its scalability. We used the RBF kernel to compute the inter-point similarity. The algorithm parameters m, p and M were set to $m = 1,000$, $p = 100$ and $M = 20,000$ respectively, and the data was input in batches of 10. Figures 5.7(a), 5.7(b) and 5.7(c) show that the proposed algorithm is linearly scalable with respect to the size and dimensionality of the data set, and almost logarithmically scalable with respect to the number of clusters, in accordance with the complexity analysis in Section 5.4.1. In Figure 5.7(a), the size of the data set is varied from $n = 100$ to $n = 10^7$, while the dimensionality and the

number of clusters are fixed at $d = 100$ and $C = 10$. The running time of the proposed algorithm increases linearly with n. Figure 5.7(b) shows the running time of the proposed algorithm as the dimensionality of the data varies between $d = 10$ and $d = 1,000$, while $n = 10^6$ and $C = 10$. Finally, the running time of our algorithm increases logarithmically as the number of clusters increases from $C = 10$ to $C = 1,000$, with $n = 10^6$ and $d = 100$, as shown in Figure 5.7(c).

5.6 Summary

In this chapter, we have proposed the sparse kernel *k*-means clustering algorithm, which can efficiently cluster large high-dimensional data sets into a large number of clusters. By sampling the data points based on their novelty, defined in terms of the statistical leverage scores, we only store the most informative points in the data, thereby limiting the memory requirements. We need to compute the kernel similarity of the data points only with respect to these sampled points, thus reducing the running time complexity. We further reduce the running time complexity by introducing sparsity into the kernel, based on the assumption that the kernel function is appropriately defined, and nearby points in the kernel space have similar labels. We demonstrated that the proposed algorithm is scalable and accurate using several large benchmark data sets.

Chapter 6

Summary and Future Work

As the amount of digital data continues to grow at a rapid rate, continued efforts to design and develop scalable and efficient algorithms to organize this data and extract useful information from it are essential. We have focused on the unsupervised learning task of clustering in this thesis. While linear clustering algorithms (e.g. k-means) are fast and scalable, they are incapable of finding the underlying clusters in real-world data sets with high accuracy. On the other hand, kernel-based clustering algorithms are accurate, but are not scalable to big data sets. We have proposed a number of kernel-based clustering algorithms which are not only scalable to data sets containing billions of data points, but also achieve cluster quality comparable to that of the existing kernel-based clustering algorithms. The proposed algorithms are primarily based on **randomly sampling** the data sets and finding the clusters using fast **iterative optimization** techniques. The main contribution of this thesis is the design of approximate algorithms for the advancement of the scalability of kernel-based clustering, while maintaining the cluster quality, and demonstrating the performance of the proposed algorithms on diverse data sets.

6.1 Contributions

The approximate batch kernel clustering algorithms proposed in Chapters 2 and 3 make the following contributions:

- The approximate kernel *k*-means algorithm in Chapter 2 demonstrates that, by using uniform random sampling, kernel-based clustering can be performed in $O(nmC + nmd)$ time, where n is the size of the given data set, d is its dimensionality, C is the number of clusters, and m is the number of samples from the data set ($m \ll n$). This running time complexity is significantly smaller than the $O(n^2C + n^2d)$ complexity of classical kernel-based clustering algorithms, given that $m \ll n$.

- In contrast to the approximate kernel *k*-means algorithm, which decomposes the kernel matrix into its low-rank components, the RFF and SV kernel clustering algorithms, introduced in Chapter 3, factor the kernel function using the Fourier transform, and project the data into a low-dimensional space spanned by the Fourier components. The RFF and SV clustering algorithms have $O(nm\log(d) + nmC)$ and $O(nm\log(d) + nC^2)$ running time complexities, respectively, where $m \ll n$ is the number of Fourier components. Both algorithms perform well on large high-dimensional data sets. The SV clustering algorithm is faster than the RFF and approximate kernel *k*-means algorithms, when the number of clusters C is small (less than 100), with a minimal loss in cluster quality.

- The error incurred by the approximate kernel *k*-means algorithm due to sampling is $O(1/m)$, which implies that the error reduces linearly, as the number of data points sampled from the data set increases. Similarly, the error incurred by the RFF and SV clustering algorithms reduces at the rate of $O(1/\sqrt{m})$ and $O(1/m)$, respectively, where m represents the number of Fourier components used for projection.

- The best clustering quality is achieved by these approximate algorithms, when the number

of samples (or the number of Fourier components) m is significantly greater than C, and the eigenvalues of the kernel matrix have a long-tailed distribution.

- The proposed algorithms achieve clustering quality similar to the kernel *k*-means and spectral clustering algorithms on large benchmark text and image data sets, containing up to 10 million data points, with significantly lower running time.

The online kernel clustering algorithms proposed in Chapters 4 and 5 make the following contributions:

- Data streams often contain unbounded number of data points, so it is impossible to store the entire data set in memory. It is also difficult to uniformly sample streaming data sets because of their arbitrary size. The approximate stream kernel *k*-means algorithm, introduced in Chapter 4, relies on *importance* sampling, and thereby uses only the most informative data points in the stream to perform clustering. Importance sampling is inherently a complex procedure because it requires the eigendecomposition of the kernel matrix. By devising an efficient online method to perform importance sampling, we have reduced its running time complexity. The approximate stream kernel *k*-means algorithm can cluster large stream data sets in $O(nd + nC)$ time.

- We have demonstrated the performance of the approximate stream kernel *k*-means on the Twitter stream. It can also be applied to find clusters in financial data, climate data, clickstreams etc.

- When the number of clusters C in the data set is large (in the order of tens of thousands), the existing kernel clustering algorithms have long running times as a result of their linear running time complexity with respect to C. By using importance sampling to sample the data set, and inducing sparsity into the kernel matrix constructed from the sampled data points, the sparse kernel *k*-means algorithm, introduced in Chapter 5, reduces this time complexity

to $O(nd + n \log C)$, with $O(1/\sqrt{m})$ approximation error, where m is the number of points sampled from the data set.

- We have demonstrated the scalability of the sparse kernel *k*-means algorithm on large heterogeneous data sets such as the Tiny image data set and the Youtube data set (text and image), containing millions of data points with upto $10,000$ clusters.

- The loss in the clustering quality by the approximate stream kernel *k*-means and the sparse kernel *k*-means algorithms is minimal when compared to the batch kernel *k*-means clustering algorithm.

The crux of the proposed algorithms is to randomly sample the large data sets and thereby, reduce the number of similarity computations required to construct the kernel matrix and cluster the data. The sample size and the sampling strategy play a crucial role in the performance of the algorithms. While the proposed batch clustering algorithms select the samples uniformly from the given data set, the online algorithms employ the more sophisticated importance sampling strategy. The importance sampling technique reduces the total number of samples required because it chooses the data points intelligently, based on the data distribution.

6.2 Future Work

Kernel-based clustering research presented in this dissertation can be further advanced as follows:

- *Parallelization.* In contrast to linear clustering algorithms, kernel-based clustering algorithms need the computation of the kernel matrix, due to which they are more difficult to parallelize than linear clustering algorithms. The approximate kernel-based clustering algorithms presented in this thesis are easier to parallelize than classical kernel-based clustering algorithms. Unlike parallel versions of the classical kernel-based clustering algorithms, which require all the data to be replicated in all the nodes, the approximate kernel-based

clustering algorithms require only the sampled data points to be replicated. This reduces the amount of memory required and the communication cost. We have demonstrated how the approximate kernel *k*-means algorithm can be executed on a distributed framework in Chapter 2. The RFF and SV clustering algorithms can be similarly parallelized. However, the remaining approximate kernel-based clustering algorithms proposed in this thesis rely on the eigenvectors of the approximate kernel matrices, and need effective online parallel techniques for eigenvector updates. Parallelization of these algorithms can aid in their deployment to large scale computing frameworks.

- *Kernel selection.* As demonstrated in Chapter 1, the kernel function used to define the inter-point similarity plays a crucial role in the efficiency and accuracy of kernel clustering. Employing the wrong kernel for clustering can adversely affect the cluster quality, and can result in clustering quality worse than that of linear clustering algorithms. However, choosing the correct kernel, and selecting the kernel parameters is a challenging task. Although a few algorithms have been proposed to learn the kernel from the data in an unsupervised manner, these algorithms have high running time complexity, resulting in their non-scalability. More scalable techniques have been developed to learn the kernel in the supervised and semi-supervised settings, but obtaining the labels for large data sets is expensive and often impossible. Development of scalable unsupervised kernel learning algorithms is a potential direction for future work.

- *Overlapping clusters.* In applications such as user community detection in social networks, users often belong to more than one community, causing the clusters to overlap with each other. Very few efforts have been made to find such overlapping clusters using kernel-based clustering techniques. Fuzzy kernel clustering techniques only compute the probability that a data point belongs to a cluster, and do not deterministically find the cluster memberships. More concrete scalable techniques need to be developed to find overlapping clusters in data.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Data Analytics. `http://searchdatamanagement.techtarget.com/definition/data-analytics`, Jan 2008.

[2] Big Data in 2020. `http://www.emc.com/collateral/analyst-reports/idc-digital-universe-2014.pdf`, Dec 2012. IDC and EMC Corp Report.

[3] M. R. Abbasifard, B. Ghahremani, and H. Naderi. A survey on nearest neighbor search methods. *International Journal of Computer Applications*, 95(25):39–52, 2014.

[4] M. E. Abbasnejad, D. Ramachandram, and R. Mandava. A survey of the state of the art in learning the kernels. *Knowledge and Information Systems*, 31(2):193–221, 2012.

[5] D. Achlioptas and F. McSherry. Fast computation of low-rank matrix approximations. *Journal of the ACM*, 54(2), 2007.

[6] M. R. Ackermann, M. Märtens, C. Raupach, K. Swierkot, C. Lammersen, and C. Sohler. StreamKM++: A clustering algorithm for data streams. *Journal of Experimental Algorithmics*, 17:1–30, 2012.

[7] R. H. Affandi, A. Kulesza, E. Fox, and B. Taskar. Nystrom approximation for large-scale determinantal processes. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pages 85–98, 2013.

[8] C. C. Aggarwal. A survey of stream clustering algorithms. In *Data Clustering: Algorithms and Applications*, pages 231–258. 2013.

[9] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for projected clustering of high dimensional data streams. In *Proceedings of the International Conference on Very Large Data Bases*, pages 852–863, 2004.

[10] N. Ailon, R. Jaiswal, and C. Monteleoni. Streaming k-means approximation. In *Proceedings of the Conference on Neural Information Processing Systems*, pages 10–18, 2009.

[11] C. Alzate and J. A. K. Suykens. Multiway spectral clustering with out-of-sample extensions through weighted kernel PCA. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(2):335–347, 2010.

[12] D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 1027–1035, 2007.

[13] K. Bache and M. Lichman. UCI machine learning repository. `http://archive.ics.uci.edu/ml`, 2013.

[14] A. Barla, F. Odone, and A. Verri. Histogram intersection kernel for image classification. In *Proceedings of the International Conference on Image Processing*, volume 3, pages 513–516, 2003.

[15] O. Beaumont, H. Larchevêque, and L. Marchal. Non-linear divisible loads: There is no free lunch. In *Proceedings of the International Parallel and Distributed Processing Symposium*, pages 1–10, 2012.

[16] H. Becker, M. Naaman, and L. Gravano. Beyond trending topics: Real-world event identification on twitter. In *Proceedings of the International AAAI Conference on Weblogs and Social Media*, pages 438–441, 2011.

[17] M. A. Belabbas and P. J. Wolfe. Spectral methods in machine learning and new strategies for very large datasets. *Proceedings of the National Academy of Sciences*, 106(2):369–374, 2009.

[18] S. Belongie, C. Fowlkes, F. Chung, and J. Malik. Spectral partitioning with indefinite kernels using the Nystrom extension. In *Proceedings of the European Conference on Computer Vision*, pages 531–542. 2002.

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

[20] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

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

[22] M. W. Berry, S. A. Pulatova, and G. W. Stewart. Computing sparse reduced-rank approximations to sparse matrices. *ACM Transactions on Mathematical Software*, 31(2):252–269, 2005.

[23] J. A. Blackard and D. J. Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and Electronics in Agriculture*, 24(3):131–152, 1999.

[24] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[25] L. Bo and C. Sminchisescu. Efficient match kernel between sets of features for visual recognition. In *Proceedings of the Conference on Neural Information Processing Systems*, pages 135–143, 2009.

[26] S. Bochner and K. Chandrasekharan. *Fourier Transforms*. Princeton University Press, 1949.

[27] L. Bottou and V. Vapnik. Local learning algorithms. *Neural Computation*, 4(6):888–900, 1992.

[28] C. Boutsidis, M. W. Mahoney, and P. Drineas. An improved approximation algorithm for the column subset selection problem. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 968–977, 2009.

[29] D. C. Brabham. *Crowdsourcing*. MIT Press, 2013.

[30] P. S. Bradley, U. M. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pages 9–15, 1998.

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

[32] F. Cao, M. Ester, W. Qian, and A. Zhou. Density-based clustering over an evolving data stream with noise. In *Proceedings of the SIAM International Conference on Data Mining*, pages 328–339, 2006.

[33] R. Cattral, F. Oppacher, and D. Deugo. Evolutionary data mining with automatic rule generalization. *Recent Advances in Computers, Computing and Communications*, pages 296–300, 2002.

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

[35] W. Chen, Y. Song, H. Bai, C. Lin, and E. Y. Chang. Parallel spectral clustering in distributed systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(3):568–586, 2011.

[36] Y. Chen and L. Tu. Density-based clustering for real-time stream data. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pages 133–142, 2007.

[37] Y. Cheng. Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):790–799, 1995.

[38] R. Chitta, A. K. Jain, and R. Jin. Sparse kernel clustering of massive high-dimensional data sets with large number of clusters. In *Proceedings of the PhD Workshop at the International Conference on Information and Knowledge Management*, 2015.

[39] 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, Department of Computer Science, Michigan State University, 2015.

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

[41] R. Chitta, R. Jin, T. C. Havens, and A. K. Jain. Scalable kernel clustering: Approximate kernel k-means. *arxiv preprint arXiv:1402.3849*, 2014.

[42] R. Chitta, R. Jin, and A. K. Jain. Efficient kernel clustering using random fourier features. In *Proceedings of the International Conference on Data Mining*, pages 161–170, 2012.

[43] R. Chitta, R. Jin, and A. K. Jain. Stream clustering: Efficient kernel-based approximation using importance sampling. In *Proceedings of the ICDM Workshop on Data Science and Big Data Analytics*, 2015.

[44] J. Chiu and L. Demanet. Sublinear randomized algorithms for skeleton decompositions. *SIAM Journal on Matrix Analysis and Applications*, 34(3):1361–1383, 2013.

[45] A. Clauset, C. R. Shalizi, and Mark E.J. Newman. Power-law distributions in empirical data. *SIAM Review*, 51(4):661–703, 2009.

[46] C. Cortes, M. Mohri, and A. Talwalkar. On the impact of kernel approximation on learning accuracy. *Journal of Machine Learning Research*, 9:113–120, 2010.

[47] T. F. Cox and M. A. A. Cox. *Multidimensional Scaling*. CRC Press, 2000.

[48] A.S. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: Scalable online collaborative filtering. In *Proceedings of the International Conference on World Wide Web*, pages 271–280, 2007.

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

[50] A. Deshpande and S. Vempala. Adaptive sampling and fast low-rank matrix approximation. In *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques*, pages 292–303. 2006.

[51] I. S. Dhillon, Y. Guan, and B. Kulis. A unified view of kernel k-means, spectral clustering and graph cuts. Technical Report TR-04-25, Department of Computer Science, University of Texas at Austin, 2004.

[52] C. Ding, X. He, and H. D. Simon. On the equivalence of nonnegative matrix factorization and spectral clustering. In *Proceedings of the SIAM Data Mining Conference*, pages 606–610, 2005.

[53] J. A. Doornik. An improved Ziggurat method to generate normal random samples. *University of Oxford*, 2005.

[54] P. Drineas, R. Kannan, and M. W. Mahoney. Fast Monte-Carlo algorithms for matrices II: Computing a low-rank approximation to a matrix. *SIAM Journal on Computing*, 36(1):158–183, 2006.

[55] P. Drineas, R. Kannan, and M. W. Mahoney. Fast Monte-Carlo algorithms for matrices III: Computing a compressed approximate matrix decomposition. *SIAM Journal on Computing*, 36(1):184–206, 2006.

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

[57] P. Drineas and M. W. Mahoney. On the Nystrom method for approximating a Gram matrix for improved kernel-based learning. *The Journal of Machine Learning Research*, 6:2153–2175, 2005.

[58] C. Eckart and G. Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[59] R. Edmonds, E. Guskin, A. Mitchell, and M. Jurkowitz. The State of the News Media 2013. `http://stateofthemedia.org/2013/newspapers-stabilizing-but-still-threatened/newspapers-by-the-numbers`, May 2013. Poynter Institute and Pew Research Center Report.

[60] A. Ene, S. Im, and B. Moseley. Fast clustering using MapReduce. In *Proceedings of the International Conference on Knowledge Discovery and Data mining*, pages 681–689, 2011.

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

[62] F. Farnstrom, J. Lewis, and C. Elkan. Scalability for clustering algorithms revisited. *ACM SIGKDD Explorations Newsletter*, 2(1):51–57, 2000.

[63] D. Feldman, M. Schmidt, and C. Sohler. Turning Big data into tiny data: Constant-size coresets for k-means, PCA and projective clustering. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 1434–1453, 2013.

[64] C. Fellbaum. *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.

[65] M. Filippone, F. Camastra, F. Masulli, and S. Rovetta. A survey of kernel and spectral methods for clustering. *Pattern Recognition*, 41(1):176–190, 2008.

[66] G. D. Forney. Generalized minimum distance decoding. *IEEE Transactions on Information Theory*, 12(2):125–131, 1966.

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

[68] C. Fraley and A. E. Raftery. How many clusters? Which clustering method? Answers via model-based cluster analysis. *The Computer Journal*, 41(8):578–588, 1998.

[69] A. Frieze, R. Kannan, and S. Vempala. Fast Monte-Carlo algorithms for finding low-rank approximations. In *Proceedings of the Foundations of Computer Science*, pages 370–378, 1998.

[70] A. Frieze, R. Kannan, and S. Vempala. Fast Monte-Carlo algorithms for finding low-rank approximations. *Journal of the ACM*, 51(6):1025–1041, 2004.

[71] J. Ginsberg, M. H. Mohebbi, R. S. Patel, L. Brammer, M. S. Smolinski, and L. Brilliant. Detecting influenza epidemics using search engine query data. *Nature*, 457(7232):1012–1014, 2008.

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

[73] A. Gittens, P. Kambadur, and C. Boutsidis. Approximate spectral clustering via randomized sketching. *arXiv preprint arXiv:1311.2854*, 2013.

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

[75] F. Godin, V. Slavkovikj, W. De Neve, B. Schrauwen, and R. Van de Walle. Using topic models for twitter hashtag recommendation. In *Proceedings of the International Conference on World Wide Web Companion*, pages 593–596, 2013.

[76] J. C. Gower. Adding a point to vector diagrams in multivariate analysis. *Biometrika*, 55(3):582–585, 1968.

[77] J. C. Gower and G. J. S. Ross. Minimum spanning trees and single linkage cluster analysis. *Applied Statistics*, pages 54–64, 1969.

[78] H. P. Graf, E. Cosatto, L. Bottou, I. Dourdanovic, and V. Vapnik. Parallel support vector machines: The cascade SVM. In *Proceedings of the Conference on Neural Information Processing Systems*, pages 521–528, 2004.

[79] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. *IEEE Transactions on Knowledge and Data Engineering*, pages 515–528, 2003.

[80] S. Guha, R. Rastogi, and K. Shim. CURE: An efficient clustering algorithm for large databases. *Information Systems*, 26(1):35–58, 2001.

[81] R. Hamid, Y. Xiao, A. Gittens, and D. DeCoste. Compact random feature maps. *arXiv preprint arXiv:1312.4626*, 2013.

[82] S. Har-Peled and S. Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 291–300, 2004.

[83] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*, volume 2. Springer, 2009.

[84] T. C. Havens. Approximation of kernel k-means for streaming data. In *Proceedings of the International Conference on Pattern Recognition*, pages 509–512, 2012.

[85] T. C. Havens and J. C. Bezdek. An efficient formulation of the improved visual assessment of cluster tendency (iVAT) algorithm. *IEEE Transactions on Knowledge and Data Engineering*, 24(5):813–822, 2012.

[86] A. Jain, Z. Zhang, and E. Y. Chang. Adaptive non-linear clustering in data streams. In *Proceedings of the International Conference on Information and Knowledge Management*, pages 122–131, 2006.

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

[88] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, Inc., 1988.

[89] A. K. Jain, R. P. W. Duin, and J. Mao. Statistical pattern recognition: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):4–37, 2000.

[90] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[91] 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 Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892, 2002.

[92] P. Kar and H. Karnick. Random feature maps for dot product kernels. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pages 583–591, 2012.

[93] G. Karypis and V. Kumar. A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. Technical report, Department of Computer Science, University of Minnesota, 1998.

[94] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley Blackwell, 2005.

[95] D. W. Kim, K. Y. Lee, D. Lee, and K. H. Lee. Evaluation of the performance of clustering algorithms in kernel-induced feature space. *Pattern Recognition*, 38(4):607–611, 2005.

[96] T. Kohonen. *Self-organizing Maps*. Springer, 2001.

[97] S. B. Kotsiantis. Supervised machine learning: A review of classification techniques. *Informatica*, 31(3), 2007.

[98] P. Kranen, I. Assent, C. Baldauf, and T. Seidl. The ClusTree: Indexing micro-clusters for anytime stream mining. *Knowledge and Information Systems*, 29(2):249–272, 2011.

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

[100] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *Proceedings of the International Conference on Computer Vision*, pages 2130–2137, 2009.

[101] A. Kumar, Y. Sabharwal, and S. Sen. A simple linear time $(1+\epsilon)$-approximation algorithm for k-means clustering in any dimensions. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 454–462, 2004.

[102] S. Kumar, M. Mohri, and A. Talwalkar. On sampling-based approximate spectral decomposition. In *Proceedings of the International Conference on Machine Learning*, pages 553–560, 2009.

[103] S. Kumar, M. Mohri, and A. Talwalkar. Sampling techniques for the Nystrom method. In *Proceedings of Conference on Artificial Intelligence and Statistics*, pages 304 – 311, 2009.

[104] T. O. Kvalseth. Entropy and correlation: Some comments. *IEEE Transactions on Systems, Man and Cybernetics*, 17(3):517–519, 1987.

[105] D. Laney. 3D data management: Controlling data volume, velocity, and variety. Technical report, META Group, 2001.

[106] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 2169–2178, 2006.

[107] Q. Le, T. Sarlos, and A. Smola. Fastfood - Approximating kernel expansions in loglinear time. In *Proceedings of the International Conference on Machine Learning*, pages 16–21, 2013.

[108] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[109] J. Lee, S. Kim, G. Lebanon, and Y. Singer. Local low-rank matrix approximation. In *Proceedings of International Conference on Machine Learning*, pages 82–90, 2013.

[110] F. Li, C. Ionescu, and C. Sminchisescu. Random Fourier approximations for skewed multiplicative histogram kernels. *Pattern Recognition*, pages 262–271, 2010.

[111] M. Li, J. T. Kwok, and B. Lu. Making large-scale Nystrom approximation possible. In *Proceedings of the International Conference on Machine Learning*, pages 631–638, 2010.

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

[113] L. L. Liu, X. B. Wen, and X. X. Gao. Segmentation for SAR image based on a new spectral clustering algorithm. *Life System Modeling and Intelligent Computing*, pages 635–643, 2010.

[114] R. Liu and H. Zhang. Sampling criteria for the Nystrom method. `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.112.6368`.

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

[116] S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[117] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

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

[119] U. Luxburg. *Clustering Stability*. Now Publishers Inc., 2010.

[120] D. MacDonald and C. Fyfe. The kernel self-organising map. In *Proceedings of the International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies*, volume 1, pages 317–320, 2002.

[121] M. Mahajan, P. Nimbhorkar, and K. Varadarajan. The planar k-means problem is NP-Hard. In *Proceedings of the International Workshop on Algorithms and Computation*, pages 274–285, 2009.

[122] M. W. Mahoney and P. Drineas. CUR matrix decompositions for improved data analysis. *Proceedings of the National Academy of Sciences*, 106(3):697–702, 2009.

[123] O. A. Maillard and R. Munos. Compressed least-squares regression. In *Proceedings of the Conference on Neural Information Processing Systems*, pages 1213–1221, 2009.

[124] S. Malinowski and R. Morla. A single pass Trellis-based algorithm for clustering evolving data streams. *Data Warehousing and Knowledge Discovery*, pages 315–326, 2012.

[125] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[126] A. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pages 169–178, 2000.

[127] G. McLachlan and D. Peel. *Finite Mixture Models*. John Wiley & Sons, 2004.

[128] M. McPherson, L. Smith-Lovin, and J. M. Cook. Birds of a feather: Homophily in social networks. *Annual Review of Sociology*, pages 415–444, 2001.

[129] A. K. Menon and C. Elkan. Fast algorithms for approximating the singular value decomposition. *ACM Transactions on Knowledge Discovery from Data*, 5(2):1–36, 2011.

[130] K. Mizumoto, H. Yanagimoto, and M. Yoshioka. Sentiment analysis of stock market news with semi-supervised learning. In *Proceedings of the IEEE/ACIS International Conference on Computer and Information Science*, pages 325–328, 2012.

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

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

[133] R. Nallapati, W. Cohen, and J. Lafferty. Parallelized variational EM for Latent Dirichlet Allocation: An experimental evaluation of speed and scalability. *ICDM Workshop on High Performance Data Mining*, pages 349–354, 2007.

[134] O. Nasraoui, C. Cardona, and C. Rojas. Using retrieval measures to assess similarity in mining dynamic web clickstreams. In *Proceedings of the International Conference on Knowledge Discovery in Data Mining*, pages 439–448, 2005.

[135] D. Newman, A. Asuncion, P. Smyth, and M. Welling. Distributed inference for Latent Dirichlet Allocation. In *Proceedings of the Conference on Neural Information Processing Systems*, pages 17–24, 2007.

[136] R. T. Ng and J. Han. CLARANS: A method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge and Data Engineering*, pages 1003–1016, 2002.

[137] N. H. Nguyen, P. Drineas, and T. D. Tran. Matrix sparsification via the Khintchine inequality. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.164.4755&rep=rep1&type=pdf`, 2009.

[138] L. Nguyen-Dinh, C. Waldburger, D. Roggen, and G. Tröster. Tagging human activities in video by crowdsourcing. In *Proceedings of the Conference on International Conference on Multimedia Retrieval*, pages 263–270, 2013.

[139] H. Ning, W. Xu, Y. Chi, Y. Gong, and T. S. Huang. Incremental spectral clustering by efficiently updating the eigen-system. *Pattern Recognition*, 43(1):113–127, 2010.

[140] L. O'Callaghan, N. Mishra, S. Guha, A. M. Meyerson, and R. Motwani. Streaming-data algorithms for high-quality clustering. In *Proceedings of the International Conference on Data Engineering*, pages 685–695, 2002.

[141] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42(3):145–175, 2001.

[142] M. Ouimet and Y. Bengio. Greedy spectral embedding. In *Proceedings of the International Workshop on Artificial Intelligence and Statistics*, pages 253–260, 2005.

[143] S. Owen, R. Anil, T. Dunning, and E. Friedman. *Mahout in Action*. Manning Publications Co., 2011.

[144] G. Petkos, S. Papadopoulos, and Y. Kompatsiaris. Two-level message clustering for topic detection in Twitter. In *Proceedings of the SNOW Data Challenge*, pages 49–56, 2014.

[145] A. K. Qinand and P. N. Suganthan. Kernel neural gas algorithms with application to cluster analysis. *Pattern Recognition*, 4:617–620, 2004.

[146] M. Raginsky and S. Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. In *Proceedings of the Conference on Neural Information Processing Systems*, pages 1509–1517, 2009.

[147] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Proceedings of the Conference on Neural Information Processing Systems*, pages 1177–1184, 2007.

[148] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis. Evaluating mapreduce for multi-core and multiprocessor systems. In *Proceedings of the IEEE Symposium on High Performance Computer Architecture*, pages 13–24, 2007.

[149] P. P. Rodrigues. Learning from ubiquitous data streams: Clustering data and data sources. *AI Communications*, 25(1):69–71, 2012.

[150] K. D. Rosa, R. Shah, B. Lin, A. Gershman, and R. Frederking. Topical clustering of tweets. In *Proceedings of the ACM SIGIR Workshop on Social Web Search and Mining*, 2011.

[151] P. J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

[152] W. Rudin. *Fourier Analysis on Groups*. Wiley-Interscience, 1990.

[153] T. Sakai and A. Imiya. Fast spectral clustering with random projection and sampling. *Machine Learning and Data Mining in Pattern Recognition*, pages 372–384, 2009.

[154] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, 2006.

[155] T. Sarlos. Improved approximation algorithms for large matrices via random projections. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 143–152, 2006.

[156] F. Schleif, A. Gisbrecht, and B. Hammer. Accelerating kernel neural gas. In *Proceedings of the International Conference on Artificial Neural Networks and Machine Learning*, pages 150–158. 2011.

[157] F. Schleif, X. Zhu, A. Gisbrecht, and B. Hammer. Fast approximated relational and kernel clustering. In *Proceedings of the International Conference on Pattern Recognition*, pages 1229–1232, 2012.

[158] B. Schölkopf, R. Herbrich, and A. Smola. A generalized representer theorem. In *Proceedings of Computational Learning Theory*, pages 416–426, 2001.

[159] B. Schölkopf and A. Smola. *Learning with kernels: Support vector machines, regularization, optimization, and beyond (Adaptive computation and machine learning)*. The MIT Press, 2001.

[160] B. Schölkopf, A. Smola, and K. R. Muller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1314, 1996.

[161] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2002.

[162] M. Shindler, A. Wong, and A. W. Meyerson. Fast and accurate k-means for large datasets. In *Proceedings of the Conference on Neural Information Processing Systems*, pages 2375–2383, 2011.

[163] H. D. Simon and H. Zha. Low-rank matrix approximation using the Lanczos bidiagonalization process with applications. *SIAM Journal on Scientific Computing*, 21(6):2257–2274, 2000.

[164] A. Smola, L. Song, and C. H. Teo. Relative novelty detection. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 5, pages 536–543, 2009.

[165] S. Steve and D. X. Zhou. Geometry on probability spaces. *Constructive Approximation*, 30:311–323, 2009.

[166] G. W. Stewart. *Matrix Perturbation Theory*. Academic Press, 1990.

[167] S. J. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. K. Chan. Cost-based modeling for fraud and intrusion detection: Results from the JAM project. In *Proceedings of the DARPA Information Survivability Conference and Exposition*, volume 2, pages 130–144, 2000.

[168] A. Strehl and J. Ghosh. Cluster ensembles - A knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3:583–617, 2003.

[169] Z. Sun and G. Fox. Study on parallel SVM based on MapReduce. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 495–561, 2012.

[170] A. Talwalkar and A. Rostamizadeh. Matrix coherence and the Nystrom method. In *Proceedings of Conference on Uncertainty in Artificial Intelligence*, 2010.

[171] P. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Pearson Education, 2007.

[172] R. Tibshirani, G. Walther, and T. Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423, 2001.

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

[174] J. W. Tukey. *Exploratory Data Analysis*. Reading, MA, 1977.

[175] J. Tzeng. Split-and-combine singular value decomposition for large-scale matrix. *Journal of Applied Mathematics*, 2013.

[176] J. K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40(4):175–179, 1991.

[177] H. Valizadegan and R. Jin. Generalized maximum margin clustering and unsupervised kernel learning. In *Proceedings of the Conference on Neural Information Processing Systems*, pages 1417–1424, 2006.

[178] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms. `http://www.vlfeat.org`, 2008.

[179] A. Vedaldi and A. Zisserman. Efficient additive kernels via explicit feature maps. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(3):480–492, 2012.

[180] S. Vega-Pons and J. Ruiz-Schulcloper. A survey of clustering ensemble algorithms. *International Journal of Pattern Recognition and Artificial Intelligence*, 25(3):337–372, 2011.

[181] R. Vidal. Subspace clustering. *IEEE Signal Processing Magazine*, 28(2):52–68, 2011.

[182] J. Wang, S. C. H. Hoi, P. Zhao, J. Zhuang, and Z. Liu. Large scale online kernel classification. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1750–1756, 2013.

[183] L. Wang, C. Leckie, R. Kotagiri, and J. Bezdek. Approximate pairwise clustering for large data sets via sampling plus extension. *Pattern Recognition*, 44(2):222–235, 2011.

[184] S. Wang and Z. Zhang. A scalable CUR matrix decomposition algorithm: Lower time complexity and tighter bound. In *Proceedings of the Conference on Neural Information Processing Systems*, pages 656–664, 2012.

[185] K. Q. Weinberger, M. Slaney, and R. Zwol. Resolving tag ambiguity. In *Proceedings of Conference on Multimedia*, pages 111–120, 2008.

[186] J. J. Whang, X. Sui, and I. S. Dhillon. Scalable and memory-efficient clustering of large-scale social networks. In *Proceedings of the International Conference on Data Mining*, pages 705–714, 2012.

[187] C. Williams and M. Seeger. Using the Nystrom method to speed up kernel machines. In *Proceedings of the Conference on Neural Information Processing Systems*, pages 682–688, 2001.

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

[189] Z. Xiaojin. Semi-supervised learning literature survey. Technical Report 1530, Department of Computer Science, University of Wisconsin-Madison, 2005.

[190] L. Xu, J. Neufeld, B. Larson, and D. Schuurmans. Maximum margin clustering. In *Advances in Neural Information Processing systems*, pages 1537–1544, 2004.

[191] D. Yan, L. Huang, and M. I. Jordan. Fast approximate spectral clustering. In *Proceedings of the International Conference on Knowledge Discovery and Data mining*, pages 907–916, 2009.

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

[193] D. Zhang, S. Chen, and K. Tan. Improving the robustness of online agglomerative clustering method based on kernel-induce distance measures. *Neural Processing Letters*, 21(1):45–51, 2005.

[194] K. Zhang and J. T. Kwok. Clustered Nystrom method for large scale manifold learning and dimension reduction. *IEEE Transactions on Neural Networks*, 21(10):1576–1587, 2010.

[195] K. Zhang, I. W. Tsang, and J. T. Kwok. Improved Nystrom low-rank approximation and error analysis. In *Proceedings of the International Conference on Machine Learning*, pages 1232–1239, 2008.

[196] R. Zhang and A. I. Rudnicky. A large scale clustering scheme for kernel k-means. *Pattern Recognition*, 4:289–292, 2002.

[197] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. *ACM SIGMOD Record*, 25(2):103–114, 1996.

[198] Y. M. Zhang, K. Huang, G. Geng, and C. Liu. Fast *k*-nn graph construction with locality sensitive hashing. *Machine Learning and Knowledge Discovery in Databases*, pages 660–674, 2013.

[199] W. Zhao, H. Ma, and Q. He. Parallel k-means clustering based on MapReduce. *Cloud Computing*, pages 674–679, 2009.

[200] J. Zhuang, J. Wang, S. C. H. Hoi, and X. Lan. Unsupervised multiple kernel learning. *Journal of Machine Learning Research*, 20:129–144, 2011.