Scalable Embeddings for Kernel Clustering on MapReduce

by

Ahmed Elgohary

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2014

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

There is an increasing demand from businesses and industries to make the best use of their data. Clustering is a powerful tool for discovering natural groupings in data. The k-means algorithm is the most commonly-used data clustering method, having gained popularity for its effectiveness on various data sets and ease of implementation on different computing architectures. It assumes, however, that data are available in an attribute-value format, and that each data instance can be represented as a vector in a feature space where the algorithm can be applied. These assumptions are impractical for real data, and they hinder the use of complex data structures in real-world clustering applications.

The kernel k-means is an effective method for data clustering which extends the k-means algorithm to work on a similarity matrix over complex data structures. The kernel k-means algorithm is however computationally very complex as it requires the complete data matrix to be calculated and stored. Further, the kernelized nature of the kernel k-means algorithm hinders the parallelization of its computations on modern infrastructures for distributed computing. This thesis defines a family of kernel-based low-dimensional embeddings that allows for scaling kernel k-means on MapReduce via an efficient and unified parallelization strategy. Then, three practical methods for low-dimensional embedding that adhere to our definition of the embedding family are proposed. Combining the proposed parallelization strategy with any of the three embedding methods constitutes a complete scalable and efficient MapReduce algorithm for kernel k-means. The efficiency and the scalability of the presented algorithms are demonstrated analytically and empirically.

Acknowledgements

Thanks to everyone who directly or indirectly helped me to complete this thesis at this quality. I would like to give a special mention to:

- My professors at Alexandria University, Prof. Noha Yousri, Prof. Mohamed Ismail, and Prof. Moustafa Youssef, for guiding my first research endeavours, for being great examples for work ethics, and for encouraging and helping me to join the University of Waterloo.
- My supervisors, Prof. Fakhri Karray and Prof. Mohamed Kamel, for the great opportunity they offered me to join the University of Waterloo, and for the freedom and trust they gave to me throughout the program.
- Mostafa Hassan for his precious advices in my first days at Waterloo.
- Waterloo professors, Prof. Ashraf Aboulnaga, Prof. Paul Marriott, and Prof. Ali Ghodsi, for their great courses that were my most valuable assist in producing this thesis.
- Ahmed Farahat for guiding me in learning about several background fundamentals, for closely mentoring all of my thesis research, and for his technical and non-technical advices.
- Radha Chitta for sharing her processed ImageNet data set that I used throughout all my experiments.
- Mike Miao for his feedback and suggestions on this work, and for presenting parts of this thesis at NIPS.
- Prof. Sagar Naik and Prof. Ali Ghodsi for accepting to be my thesis readers.

Dedication

This thesis is dedicated to my parents, Iman and Ali.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

In today's era of big data, there is an increasing demand from businesses and industries to get an edge over competitors by making the best use of their data. Clustering is one of the powerful tools that data scientists can employ to discover natural groupings in data. The k-means algorithm [1] is the most commonly-used data clustering method. It has gained popularity for its effectiveness on many data sets as well as the ease of its implementation on different computing architectures. The k-means algorithm, however, assumes that data are available in an attribute-value format, and that all attributes can be turned into numeric values so that each data instance is represented as a point or vector in some feature space where the algorithm can be applied. These assumptions are impractical for real data and they hinder the utilization of complex data structures in real-world clustering problems. Examples include grouping users in a social network based on their friendship networks, clustering customers according to their behaviour, and grouping proteins based on their structures. Data scientists tend to simplify these complex structures to a vectorized format, and would accordingly lose the richness of the data they have.

In order to overcome this problem, much research has been conducted on clustering algorithms that work on similarity matrices over data instances, rather than on a vector representation of the data in a feature space. This has led to the advance of different similarity-based methods for data clustering, such as kernel k-means [2] and spectral clustering [3]. The focus of this thesis is on the kernel k-means algorithm [2]. Different from the traditional k-means algorithm, the kernel k-means algorithm works on kernel matrices which encode different aspects of similarity between complex data structures [4]. It has also been shown that the widely-accepted spectral clustering method has an objective function which is equivalent to a weighted variant

of the kernel k-means algorithm [2], which means that optimizing that criterion allows for an efficient implementation of the spectral clustering algorithm, in which computationally complex eigendecomposition step is bypassed [5]. Accordingly, the methods proposed in this thesis can be leveraged for scaling the spectral clustering method on MapReduce.

The kernel k-means algorithm, however, depends on calculating and storing the kernel matrix over all data instances. Further, all entries of the kernel matrix need to be accessed in each iteration. As a result, the kernel k-means algorithm has quadratic space and time complexities per iteration. These complexities become scalability bottlenecks as the dataset size increases. Some recent work [6, 7] has been proposed to approximate the kernel k-means clustering, and allow its application to large data. However, those algorithms are designed for centralized settings, and assume that the data will fit on the memory/disk of a single machine.

This thesis proposes a family of algorithms for scaling the kernel k-means over cloud infrastructures for distributed computing. Such infrastructures tend to be composed of several commodity nodes, each of which is of a limited memory and computing power [8–10]. The nodes are connected together in a shared-nothing cluster, which means that data transfers between different nodes are done through the network. In such settings, ensuring the scalability and fault tolerance of data analysis tasks is troublesome. MapReduce [9] is a programming model, supported by an execution framework that provides scalable and fault-tolerant execution of analytical data processing tasks over distributed infrastructures of commodity nodes. The proposed algorithms in this thesis are designed to perfectly fit into the MapReduce programming model, and to adhere to its computational constraints. We also optimize the execution of the proposed algorithms by considering the different performance aspects of the target computing infrastructure.

Our approach is based on eliminating the scalability bottlenecks of the kernel k-means by first learning an embedding of the data instances, and then using this embedding to approximate the cluster assignment step in each iteration of the kernel k-means algorithm. We show that this approach leads to a unified and MapReduce-efficient scaling strategy. In addition, we generalize our approach by defining a family of embeddings characterized by only four properties, which ensure the correctness of any embedding method in the defined family for scaling the kernel k-means on MapReduce.

1.2 Summary of Contributions

The contributions of the thesis can be summarized as follows.

- The thesis proposes a generic family of kernelized low-dimensional embeddings, which is called Approximate Nearest Centroid (APNC) embeddings, and defines its computational and statistical properties that facilitate scaling kernel k-means on MapReduce.
- Exploiting the properties of APNC embeddings, the thesis presents a unified and efficient parallelization strategy on MapReduce for approximating the kernel k-means using any APNC embedding.
- The thesis proposes three practical instances of APNC embeddings which are based on the Nyström method and the use of p-stable distributions for approximating vector norms.
- The presented algorithms are analyzed in terms of their space, time, and network communication complexities. These analytical results are used to prove the efficiency and the the scalability of the proposed approach.
- Extensive medium and large-scale experiments were conducted to compare the proposed approach to state-of-the-art kernel k-means approximations, and to demonstrate the effectiveness and scalability of the presented algorithms.

1.3 Thesis Organization

The rest of this chapter describes the notations used throughout the thesis. The first part of Chapter 2 gives an introductory background on data clustering, kernel methods, and cloud analytics using MapReduce, while the second part discusses related work on scaling kernel-based clustering methods, and the recent efforts at adopting data anlytics tasks on MapReduce. The proposed approach and algorithms are given in Chapter 3. The experiments conducted, and their results, are described in Chapter 4. Finally, the conclusion of the thesis, and a set of possible future extensions are presented in Chapter 5.

1.4 Notations

The following notations are used throughout the thesis unless otherwise indicated. Scalars are denoted by small letters (e.g., m, n), sets are denoted in script letters (e.g., $\mathcal{L}$), vectors are denoted by small bold italic letters (e.g., $\boldsymbol{\phi}$, $\boldsymbol{y}$), and matrices are denoted by capital letters (e.g., Φ, Y). In addition, the following notations are used:

For a set $\mathcal{L}$:

$\mathcal{L}^{(b)}$	the subset of $\mathcal{L}$ corresponding to the data block b.
$\lvert\mathcal{L}\rvert$	the cardinality of the set.

For a vector $\boldsymbol{x} \in \mathbb{R}^m$:

$\boldsymbol{x}_i$	i-th element of $\boldsymbol{x}$.
$\boldsymbol{x}^{(i)}$	the i-th vector.
$\boldsymbol{x}_{[b]}$	the vector $\boldsymbol{x}$ corresponding to the data block b.
$\lVert\boldsymbol{x}\rVert_p$	the ℓ_p-norm of $\boldsymbol{x}$.

For a matrix $A \in \mathbb{R}^{m \times n}$:

A_{ij}	(i, j)-th entry of A.
$A_{i:}$	i-th row of A.
$A_{:j}$	j-th column of A.
$A_{\mathcal{L}:}$, $A_{:\mathcal{L}}$	the sub-matrices of A which consist of the set $\mathcal{L}$ of rows and columns respectively.
A^T	the transpose of A.
$A^{(b)}$	the sub-matrix of A corresponding to the data block b.

Chapter 2

Background and Related Work

2.1 MapReduce Framework

Distributed cloud computing infrastructures tend to be composed of several commodity nodes, each of which is of a limited memory and computing power [8–10]. The nodes are connected together in a shared-nothing cluster which, means that data transfers between different nodes are done through the network. In such settings of infrastructure, ensuring the scalability and fault tolerance of data analysis tasks is troublesome. MapReduce [9] is a programming model supported by an execution framework that provides scalable and fault tolerant execution of analytical data processing tasks over distributed infrastructures of commodity nodes. The simplicity of the MapReduce API, together with its scalable and fault-tolerant execution framework, distinguished MapReduce and its open-source implementation Hadoop [11] as the most attractive paradigm for data analytics tasks on large-scale cloud computing infrastructures.

The rationale behind MapReduce is to impose a set of constraints on data access at each node and communication between different nodes, to ensure both the scalability and fault-tolerance of the analytical tasks. A MapReduce job is executed in two phases of user-defined data transformation functions, namely, the *map* and *reduce* phases. The input data is split into physical blocks distributed across the nodes, each block is viewed as a list of key-value pairs. In the first phase, the key-value pairs of each input block b are processed by a single *map* function, running independently on the node where the block b is stored. The key-value pairs are provided one-by-one to the *map* function; the output of the *map* function is another set of intermediate key-value pairs. The values associated with the same key across all nodes are grouped together and provided as an input to the *reduce* function in the second phase. Different groups of values are processed in parallel on different machines. The output of each *reduce* function is a third set of key-value

pairs, and collectively considered the output of the job. For complex analytical tasks, multiple jobs are typically chained together [12], and/or many rounds of the same job are executed on the input data set [13].

It is important to note that in addition to the processing time of the *map* and *reduce* functions, a major portion of the job execution time is that taken to move the intermediate key-value pairs across the network. Hence, minimizing the size of the intermediate key-value pairs significantly reduces the overall running time of MapReduce jobs. Further, since the individual nodes in cloud computing infrastructures are of very limited memory, a scalable MapReduce-algorithm should ensure that the memory required per node remains within the bound of commodity memory sizes as the data size increases. A significant amount of research has been devoted for scaling complex data analytics algorithms on MapReduce by developing efficient parallelization strategies, or even by introducing novel approximations that lead to MapReduce-efficient algorithms. Such algorithms spanned text mining [14], graph mining [15, 16], nonnegative matrix factorization [17], feature selection [18], regression [19], PageRank [20] and most recently column subset selection [12].

2.2 Data Clustering

Data clustering is an unsupervised learning task that aims at discovering natural grouping in unlabelled input datasets. Over the past decades, researchers have developed various application-specific [21, 22] or general-purpose [23, 24] data clustering approaches. Data clustering has been used in a wide spectrum of applications, ranging from news organization [20] to indoor localization [25].

The k-means algorithm [1] is the most widely used algorithm for data clustering. The objective of the algorithm is to group the data points into k clusters, such that the Euclidean distances between data points in each cluster and that cluster's centroid are minimized. Let $\mathcal{P}_c$ denote the set of data instances assigned to the cluster c and $\bar{x}^{(c)}$ denote the centroid of the data instances in $\mathcal{P}_c$ (i.e. $\bar{x}^{(c)} = \frac{1}{n_c} \sum_{i \in \mathcal{P}_c} x^{(i)}$) where $n_c = |\mathcal{P}_c|$. The k-means objective is to assign the input data points to k disjoint sets $\mathcal{P}_c$ for $c = 1, 2, \ldots, k$ such that following loss function is minimized:

$$Loss = \sum_{c=1}^{k} \sum_{i \in \mathcal{P}_c} \left\| x^{(i)} - \bar{x}^{(c)} \right\|_2^2 . \tag{2.1}$$

An iterative algorithm, namely Lloyd's algorithm [26], is usually used for the optimization of the loss function in 2.1. In each iteration, Lloyd's algorithm assigns each data point to the

Algorithm 1 Lloyd k-means

Input: Dataset $\mathcal{X} = \{x^{(1)}, x^{(2)}, \ldots, x^{(n)}\}$, Number of clusters k
Output: k clusters $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k$

Generate initial k centroids $\bar{x}^{(1)}, \bar{x}^{(2)}, \ldots, \bar{x}^{(k)}$
repeat until convergence
    Reset all cluster sets $\mathcal{P}_c \leftarrow \{\}$, for $c = 1, \ldots, k$
    for $i = 1 : n$
        $\hat{c} \leftarrow \arg\min_c \left\| x^{(i)} - \bar{x}^{(c)} \right\|_2^2$
        $\mathcal{P}_{\hat{c}} \leftarrow \mathcal{P}_{\hat{c}} \cup x^{(i)}$
    end
    Update cluster centroids as $\bar{x}^{(c)} = \frac{1}{n_c} \sum_{x \in \mathcal{P}_c} x$, for $c = 1, \ldots, k$
end

nearest centroid, and calculates new centroids based on the current assignment of the data points. Afterwards, cluster centroids are updated based on the new cluster assignments. Algorithm 1 outlines the steps of Lloyd's algorithm. The algorithm is known to converge to a local-optimal solution. In addition, it can be observed from the steps of algorithm 1 that data instances are required to be represented in a vector form. Further, the k-means objective in general forces the clusters to be separated by a hyperplane [5], which makes the k-means unable to discover non-linearly separable clusters.

2.3 Kernel Methods

One appealing idea for dealing with data of non-linear structures is to use a mapping (embedding) function to map each data instance x to a typically high-dimensional feature space (as $\phi = \varphi(x)$) in which data instances become linearly separable. Applying a linear learning algorithm (e.g. k-means) to the data instances in the new feature space provides an elegant replacement for developing non-linear algorithms and applying them directly to non-linearly separable data in their original form. However, mapping each data instance to the new feature space might be intractable, or a computationally expensive step. Further, to obtain the desired linear-separability properties, the new feature space is usually extremely high dimensional (possibly of infinite dimensionality) and accordingly, working with the explicit mappings ϕ is most often infeasible.

Mercer's theorem [27] was the key result that enabled researchers to put the idea of mapping data instances to high-dimensional spaces into a computationally tractable framework, known as

kernel methods [4]. Mercer's theorem indicates that a symmetric function (a kernel) $\kappa(.,.)$ can be expressed as an inner producted $\kappa(x^{(i)}, x^{(j)}) = \varphi(x^{(i)})^T \varphi(x^{(j)})$ for some mapping function φ if and only if $\kappa(.,.)$ is positive semi-definite. That is, the matrix whose entries are $[\kappa(x^{(i)}, x^{(j)})]_{i,j}$ is positive semi-definite for any set $\mathcal{X} = \{x^{(1)}, x^{(2)}, \ldots, x^{(n)}\}$. According to that theorem, if each data access step of a linear learning algorithm is expressed as an inner product of a pair of instances, one will no longer need to explicitly compute each mapping $\varphi(x^{(i)})$. Instead, each inner product $\varphi(x^{(i)})^T \varphi(x^{(j)})$ can be substituted with the output of the kernel function between the two corresponding original data instances $\kappa(x^{(i)}, x^{(j)})$. Throughout this thesis, for an input dataset $\mathcal{X} = \{x^{(1)}, x^{(2)}, \ldots, x^{(n)}\}$, we refer to the matrix $[\kappa(x^{(i)}, x^{(j)})]_{i,j}$ as the *kernel matrix*. The new feature space endowed by a kernel function is referred to as the *kernel space*.

The idea of kernel methods have been applied to several machine learning algorithms. The most popular example of such an algorithm is the support vector machines [4]. Commonly used kernel functions that have been employed successfully in various learning tasks are (1) the Radial Basis Function (RBF) kernel, which is given by:

$$\kappa(x^{(i)}, x^{(j)}) = \exp\left(-\frac{\left\|x^{(i)} - x^{(j)}\right\|^2}{2\sigma^2}\right),$$

where σ is a tuning parameter known as the kernel width; (2) the Polynomial kernel, which is given by:

$$\kappa(x^{(i)}, x^{(j)}) = \left({x^{(i)}}^T x^{(j)} + 1\right)^d,$$

where d is a tuning parameter that determines the polynomial degree; and (3) the Sigmoid kernel, which is given by

$$\kappa(x^{(i)}, x^{(j)}) = tanh\left(a{x^{(i)}}^T x^{(j)} + b\right),$$

where both a and b are tuning parameters.

Kernel methods have also been shown to be useful for handling datasets of complex structures [28]. Examples of such data structures are text strings, graphs, and biological sequences. Since most learning algorithms work on data points represented in a vector space, data scientists had to simplify these complex structures to a vectorized format and accordingly lose the richness of the data they had. Kernel methods allow for working with such complex data structures implicitly, if the "kernelized" learning algorithm is provided with a proper kernel function that encodes the similarity between each pair of data instances.

2.4 Kernel Approximations

For a dataset of size n, computing the kernel matrix is of $\mathcal{O}(n^2)$, and storing that kernel matrix requires $\mathcal{O}(n^2)$ of memory. Those quadratic complexities hinder the application of kernel methods to large-scale datasets. In the past few years, methods for kernel approximation have been introduced to enable using kernel methods for such datasets. The two most popular methods for kernel approximations are the Nyström approximation [29] and the random Fourier features [30] [31].

2.4.1 The Nyström Approximation

The Nyström method [32] provides a low-rank approximation of a kernel matrix K of n data instances, using the kernel matrix between all data instances and a few set of data instances $\mathcal{L}$ as

$$\tilde{K} = D^T A^{-1} D \,, \tag{2.2}$$

where $|\mathcal{L}| = l \ll n$, $A \in \mathbb{R}^{l \times l}$ is the kernel matrix over the data instances in $\mathcal{L}$, and $D \in \mathbb{R}^{l \times n}$ is the kernel matrix between the data instances in $\mathcal{L}$ and all data instances. The approximation in 2.2 can be derived as follows: let Φ be a $d \times n$ matrix of the n data instances mapped to the kernel space (i.e. $K = \Phi^T \Phi$), and $\Phi_{:\mathcal{L}}$ be a $d \times l$ sub-matrix of Φ whose columns are the data instances in $\mathcal{L}$. Suppose each column in Φ is approximated as a linear combination of the columns of $\Phi_{:\mathcal{L}}$, i.e. an approximate data matrix $\tilde{\Phi}$ is given by $\tilde{\Phi} = \Phi_{:\mathcal{L}} T$. The optimal value for T obtained by minimizing the least square error $\|\Phi - \Phi_{:\mathcal{L}} T\|_2^2$ is $T = (\Phi_{:\mathcal{L}}^T \Phi_{:\mathcal{L}})^{-1} \Phi_{:\mathcal{L}}^T \Phi$. Substituting this value of T back to $\tilde{\Phi}$ and computing the approximate kernel matrix as $\tilde{K} = \tilde{\Phi}^T \tilde{\Phi}$ yields the approximation in 2.2 [33].

For a fixed value of l, the accuracy of the Nyström approximation is determined by the set of samples $\mathcal{L}$. A number of approaches have been proposed in the literature for selecting the samples used to compute the approximation. These approaches have varied from "one-shot" uniform and non-uniform probabilistic sampling [34], to adaptive probabilistic [35] and deterministic sampling [36].

The ensemble Nyström method [37] is an extension of the Nsyström approximation given in Eq. 2.2, in which a low rank approximation of the kernel matrix is given by a weighted sum of q low rank approximations computed using disjoint subsets of samples as

$$\tilde{K} = \sum_{b=1}^{q} \mu_b \tilde{K}^{(b)} \tag{2.3}$$

where each $\tilde{K}^{(b)}$ is referred to as an expert, and is computed using a small subset of samples $\mathcal{L}^{(b)}$ as defined in Eq. 2.2 as

$$\tilde{K}^{(b)} = {D^{(b)}}^T {A^{(b)}}^{-1} D^{(b)} \tag{2.4}$$

where $\left|\mathcal{L}^{(b)}\right| = l^{(b)} \ll n$, $A^{(b)} \in \mathbb{R}^{l^{(b)} \times l^{(b)}}$ is the kernel matrix over the data instances in $\mathcal{L}^{(b)}$, and $D^{(b)} \in \mathbb{R}^{l^{(b)} \times n}$ is the kernel matrix between the data instances in $\mathcal{L}^{(b)}$ and all data instances. The authors in [37] proved that computing a low-rank approximation for a given kernel matrix using the ensemble Nyström method with $q > 1$ achieves a better approximation than the Nyström method, when the same number of sample data instances is used.

Three methods for computing the weights μ_b have been discussed in [37]. In the uniform-weights method, all weights are assigned the same value as $\mu_b = \frac{1}{q}$ for $b = 1, 2, .., q$. The two other methods are based on the use of a validation sample of data instances, whose exact pairwise kernel matrix is denoted as K_V. In the exponential-weights methods, the weight corresponding to each expert b is computed as $\mu_b = \exp(-\eta\epsilon_b)/Z$, where ϵ_b is the approximation error (measured in terms of the Frobenius norm or the spectral norm) of estimating K_V using the expert b, η is a tuning parameter, and Z is a normalization term. The third method is based the solving of a regression problem to find the set of weights the minimizes the least square error of estimating K_V by combining all the experts in Eq. 2.3.

As the value of l increases, the $\mathcal{O}(l^3)$ computational complexity of computing the inverse of the matrix A can be an expensive step of computing the Nyström approximation given by Eq. 2.2. Li *et al.* [38] suggested to reduce that complexity by computing an approximate A^{-1} using the stochastic singular value decomposition (SSVD) presented in [39]. That approximation was shown to significantly reduce the computational complexity of the Nyström approximation with a quite minor effect on the accuracy of the computed low-rank kernel approximation. Clearly, the same approximation can be extended to be applied to the ensemble Nyström method, where each of ${A^{(b)}}^{-1}$ is approximated using the SSVD.

2.4.2 Random Fourier Features

Rahimi and Recht have shown in [30] that any shift-invariant kernel (i.e. kernels in the form $\kappa(\boldsymbol{x}, \boldsymbol{y}) = \kappa(\boldsymbol{x} - \boldsymbol{y})$) can be approximated as

$$\kappa(\boldsymbol{x} - \boldsymbol{y}) = \mathbb{E}_{\boldsymbol{\omega}} \left[g(\boldsymbol{\omega}, \boldsymbol{x})^T g(\boldsymbol{\omega}, \boldsymbol{y}) \right] ,$$

where

$$g(\boldsymbol{\omega}, \boldsymbol{y}) = \left[cos(\boldsymbol{\omega}^T \boldsymbol{x}) \;\; sin(\boldsymbol{\omega}^T \boldsymbol{x}) \right]^T$$

and $\boldsymbol{\omega}$ follows a distribution $p(\boldsymbol{\omega})$ that is determined by $\kappa(.,.)$. By approximating the expectation above by the sample mean using m i.i.d. vectors drawn from $p(\boldsymbol{\omega})$, each of which is denoted as $\omega^{(i)}$, $\kappa(\boldsymbol{x}-\boldsymbol{y})$ can approximately be expressed in the form of an inner producted as

$$\kappa(\boldsymbol{x}-\boldsymbol{y}) \approx z(x)^T z(y)$$

where

$$z(x) = \frac{1}{\sqrt{m}} \left[cos({\omega^{(1)}}^T x), \ldots, cos({\omega^{(m)}}^T x), sin({\omega^{(1)}}^T x), \ldots, sin({\omega^{(m)}}^T x) \right] . \tag{2.5}$$

Thus, computing the full kernel matrix of a given dataset can be replaced by computing the corresponding explicit mapping $z(x)$ of each data instance x, and applying the linear learning algorithm directly to the computed representations.

The advantage of the random Fourier features (RFF) approach is that it enables exploiting the ability of kernel methods to handle non-linearly separable data using a computationally inexpensive linear learning algorithm. However, it can be seen from Eq. 2.5 that the RFF approach requires all data instances to be in a vectorized form, which hinders employing the RFF approximation for datasets of complex structures. In addition, the RFF approach is still limited to shift-invariant kernels. Recent work has shown how to extend the idea of using explicitly computed embeddings to approximate other types of kernels, such as additive homogenous kernels [40] and dot product kernels [41]. However, all of these approaches require data instances to be vectorized, and they are still limited to specific types of kernels.

2.5 Kernel-Based Clustering

This Section describes the two most popular and commonly-used kernel-based clustering algorithms. The first is the kernel k-means algorithm [2] which is the result of applying the idea of kernel methods to the k-means described in Section 2.2. The second is the spectral clustering algorithm [3] which in principle works on arbitrary pairwise similarity matrices (i.e. the similarity function used does not have to adhere to Mercer's definition). However, we list the spectral clustering under this section as it was shown to have an equivalent objective function to that of a weighted-variant of the kernel k-means [5]. Accordingly, the proposed kernel k-means algorithms in this thesis can possibly be extended for scaling spectral clustering algorithms on MapReduce.

2.5.1 Kernel k-Means

The kernel k-means [2] is a variant of the k-means algorithm in which the idea of kernel methods is used to enable the algorithm to work on datasets of complex structures and/or discover non-linearly separable clusters. Let $\boldsymbol{\phi}^{(i)}$ be the mapping of a data instance i in the feature space endowed implicitly by the kernel function $\kappa(.,.)$. The k-means loss function in Eq. 2.1 is extended in the kernel k-means to be

$$Loss = \sum_{c=1}^{k} \sum_{i \in \mathcal{P}_c} \left\| \boldsymbol{\phi}^{(i)} - \bar{\boldsymbol{\phi}}^{(c)} \right\|_2^2 , \tag{2.6}$$

where $\bar{\boldsymbol{\phi}}^{(c)} = \frac{1}{n_c} \sum_{i \in \mathcal{P}_c} \phi^{(i)}$ and $n_c = |\mathcal{P}_c|$.

In Lloyd's Algorithm (Algorithm 1), cluster assignments are made based on the ℓ_2-distance between $\boldsymbol{\phi}^{(i)}$ and each cluster centroid $\bar{\boldsymbol{\phi}}^{(c)}$ as

$$\pi(i) = \arg\min_{c} \left\| \boldsymbol{\phi}^{(i)} - \bar{\boldsymbol{\phi}}^{(c)} \right\|_2 . \tag{2.7}$$

Since neither $\boldsymbol{\phi}^{(i)}$ nor $\bar{\boldsymbol{\phi}}^{(c)}$ can be assumed to be accessible explicitly, the square of the ℓ_2-distance in Eq. (2.7) is expanded in terms of entries from the kernel matrix K as:

$$\left\| \boldsymbol{\phi}^{(i)} - \bar{\boldsymbol{\phi}}^{(c)} \right\|_2^2 = K_{ii} - \frac{2}{n_c} \sum_{a \in \mathcal{P}_c} K_{ia} + \frac{1}{n_c^2} \sum_{a,b \in \mathcal{P}_c} K_{ab} , \tag{2.8}$$

where K_{ab} is the (a,b)-th entry of the kernel matrix.

That expansion makes the computational complexity of finding the nearest centroid to each data instance $\mathcal{O}(n)$, and that of a single iteration over all data instances $\mathcal{O}(n^2)$. Further, an $\mathcal{O}(n^2)$ space is needed to store the kernel matrix K. These quadratic complexities hinder applying the kernel k-means algorithm to large datasets if the clustering is performed on a single node. In MapReduce settings, computing the kernel matrix requires $\mathcal{O}(n^2)$ data transfers across the network. In addition, to find the best cluster assignment for a data instance i, the kernel between i and all the other data instances $K_{:i}$ must be loaded in the memory of a single node, which incurs an $\mathcal{O}(n)$ memory requirement per node. As the dataset size n increases, $K_{:i}$ will not fit into the memory of a single commodity node. Accordingly, it can be concluded that the original kernel k-means algorithm cannot be implemented on MapReduce in a scalable way.

2.5.2 Spectral Clustering

Spectral clustering is derived from formulating the clustering problem as a normalized graph partitioning problem, where the vertices of the graph correspond to data instances and each edge between two vertices corresponds to the similarity between the two data instances represented by the two vertices. The objective of the spectral clustering algorithm is to minimize the *graph cut* resulting from partitioning the nodes into disjoint groups. Several cut objectives have been presented in previous works [42–44]. The most commonly used one is the normalized cut [43, 45], which is defined in the case of bi-partitioning the graph into two subgraphs $\mathcal{A}$ and $\mathcal{B}$ as

$$NCut(\mathcal{A},\mathcal{B}) = cut(\mathcal{A},\mathcal{B})\left(\frac{1}{vol(\mathcal{A})}+\frac{1}{vol(\mathcal{B})}\right), \tag{2.9}$$

where $cut(\mathcal{A},\mathcal{B})$ is the sum of all edges connecting vertices in $\mathcal{A}$ and vertices in $\mathcal{B}$, and $vol(\mathcal{A})$ is the sum of the edges connecting all pairs of vertices in $\mathcal{A}$.

The spectral clustering algorithm requires computing an $n \times n$ matrix of all pairwise similarities A of the input data instances (graph adjacency matrix). In addition, the algorithm requires finding the eigenvectors of the $n \times n$ Laplacian matrix corresponding to the computed adjacency matrix, which makes the time complexity of the spectral clustering algorithm $\mathcal{O}(n^3)$, in addition to the $\mathcal{O}(n^2)$ space needed to store the laplacian matrix. The final clustering is obtained by running the k-means algorithm on the k leading eigenvectors of the Laplacian matrix, where k is the number of the desired clusters.

Multiple approximations have been presented in previous work [45–47] to tackle the scalability limitations of the spectral clustering. These approximations adopted different sample-based approaches, in which l ($l \ll n$) data points are used to approximate the spectral clustering. As a result, one has only to compute an eigen-decomposition of an $l \times l$ matrix which obviates the $\mathcal{O}(n^3)$ of the original spectral clustering algorithm, and replaces it with the $\mathcal{O}(l^3)$ complexity required for decomposing the $l \times l$ matrix. In [45], Fowlkes *et al.* exploited the the Nyström approximation outlined in section 2.4.1 to compute approximate k-leading eigenvectors of the Laplacian matrix of the given dataset, which reduces the complexity of the spectral clustering algorithm to $\mathcal{O}(l^3 + lnt)$, where t is the required number of the k-means iterations needed for the algorithm to converge. In [47], Yan *et al.* suggested to find the exact clustering of l data points (obtained via the k-means clustering or by random projections) using the original spectral clustering algorithm. The obtained clustering labels are then propagated to the entire dataset. That approach was shown to reduce the computational complexity of the spectral clustering to $\mathcal{O}(l^3 + lnt)$ too. Later in [46], Chen and Cai presented another approximation which was shown empirically to achieve superior clustering accuracy to that of the Nyström-based approximation and the approximation of Yan *et al.* [47]. The approach of Chen and Cai [46] is based on finding

a sparse representation of the entire dataset based on a sample of l data points (drawn randomly out of the given dataset or computed using the k-means clustering). That approach reduces the runtine complexity of the spectral clustering to $\mathcal{O}(l^3 + l^2 n + lnt)$.

2.5.3 Equivalence of Kernel k-Means and Spectral Clustering

The weighted kernel k-means is a variant of the kernel k-means, in which each data instance i is assigned a weight w_i [5]. The loss function in 2.6 is accordingly modified to be

$$Loss = \sum_{c=1}^{k} \sum_{i \in \mathcal{P}_c} w_i \left\| \boldsymbol{\phi}^{(i)} - \bar{\boldsymbol{\phi}}^{(c)} \right\|_2^2 , \tag{2.10}$$

where

$$\bar{\boldsymbol{\phi}}^{(c)} = \frac{\sum_{i \in \mathcal{P}_c} w_i \boldsymbol{\phi}^{(i)}}{\sum_{i \in \mathcal{P}_c} w_i} .$$

Dhillon *et al.* [5] showed that different cut objectives of the spectral clustering algorithm have corresponding similarly weighted kernel k-means objective functions. For instance, the weighted kernel k-means variant corresponding to the normalized graph cut defined in 2.9 is obtained by setting the weight of each data instance w_i to the degree of the corresponding vertex in the (i.e. $w_i = \sum_{j=1}^{n} A_{ij}$), and the kernel matrix K to $K = \sigma D^{-1} + D^{-1} A D^{-1}$, where D is a diagonal matrix whose diagonal entries are the degrees of the corresponding vertices, and σ is a positive scalar that ensures the positive semi-definiteness of the kernel matrix K. The authors in [5] exploited that equivalence to propose an approximate spectral clustering approach that eliminates the $\mathcal{O}(n^3)$ complexity incurred when finding the leading eigenvectors of the graph Laplacian in the original spectral clustering algorithm.

2.6 Related Work

2.6.1 Kernel k-Means Approximations

The quadratic runtime complexity per iteration, in addition to the quadratic space complexity of the kernel k-means, have limited its applicability to even medium-scale data sets on a single machine. Recent work [6,7] to tackle these scalability limitations has focused only on centralized settings with the assumption that the data set being clustered fits into the memory/disk of a single machine. In specific, Chitta *et al.* [6] suggested restricting the clustering centroids to an at most

rank-l subspace of the span of the entire data set where $l \ll n$. That approximation reduces the runtime complexity per iteration to $\mathcal{O}(l^2k + nlk)$, and the space complexity to $\mathcal{O}(nl)$, where k is the number of clusters. However, that approximation is not sufficient for scaling kernel k-means on MapReduce, since assigning each data point to the nearest cluster still requires accessing the current cluster assignment of all data points. It was also noticed by the authors that their method is equivalent to applying the original kernel k-means algorithm to the rank-l Nyström approximation of the entire kernel matrix [6]. That is algorithmically different from the Nyström-based embedding method proposed in this thesis (details in Section 3.3) in the sense that we use the concept of the Nyström approximation to learn low-dimensional embedding for all data instances which allows for clustering the data instances by applying a simple and MapReduce-efficient algorithm on their corresponding embeddings.

Later, Chitta *et al.* [7] exploited the Random Fourier Features (RFF) approach [30] to propose fast algorithms for approximating the kernel k-means. However, these algorithms inherit the limitations of the used RFF approach, as discussed in Section 2.4.2. Furthermore, the theoretical and empirical results of Yang *et al.* [31] showed that the kernel approximation accuracy of RFF-based methods depends on the properties of the eigenspectrum of the original kernel matrix, and ensuring acceptable approximation accuracy requires using a large number of Fourier features, which increases the dimensionality of the computed RFF-based embeddings. In our experiments, we empirically show that our kernel k-means methods achieve clustering accuracy superior to those achieved using the state-of-the-art approximations presented in [6] and [7].

2.6.2 Distributed Data Clustering

Clustering distributed data on infrastructures other than MapReduce has been considered significantly in previous work [48–52]. Forman and Zhang. [51] focused on centroids-based clustering algorithms such as the k-means. Their approach is based on sharing the same set of centroids among all nodes, where each node uses these centroids to compute sufficient statistics of small size about the cluster assignments made locally. The sufficient statistics computed at all nodes are grouped at the end of each iteration at a centralized server that computes updated centroids to be used in the following iteration. Januzaj *et at.* [50] proposed clustering the data locally at each node once, and extracting representatives out of the resulting clusters. The local representatives are sent to a centralized server which computes a set of global representatives that are then shared among all nodes to compute the final cluster assignment of each data point. More recently, Datta *et al.* [49] presented a fully decentralized approach to clustering distributed data over a peer-to-peer network using the k-means algorithm. The basic idea behind their approach is that each node runs a single k-means iteration over its local data then, the resulting local centroids are synchronized only with the neighbour nodes at the end of each iteration. Afterwards, each node

starts the next iteration using the synchronized centroids until satisfying a convergence criteria. The authors also showed how the algorithm should behave in a dynamic networks, where the network structure or the data change over time. Later, Elgohary and Ismail [48] extended the approach of [49] with an online cluster assignment approach that can achieve better clustering accuracy, and most importantly, prevents ending up with one or more empty clusters.

2.6.3 MapReduce for Kernel Clustering

Other than the kernel k-means, the spectral clustering algorithm [3] is considered a powerful approach to kernel-based clustering. Chen *et al.* [53] presented a distributed implementation of the spectral clustering algorithm using an infrastructure composed of MapReduce, MPI [54], and SStable[1]. In addition to the limited scalability of MPI, the reported running times are very large. We believe this was mainly due to the very large network overhead resulting from building the kernel matrix using SSTable. Later, Gao *et al.* [55] proposed an approximate distributed spectral clustering approach that relied solely on MapReduce. The authors showed that their approach significantly reduced the clustering time compared to that of Chen *et al.*, [53]. However, in the approach of Gao *et al.* [55], the kernel matrix is approximated as a block-diagonal, which enforces inaccurate pre-clustering decisions that could result in degraded clustering accuracy.

Scaling other algorithms for data clustering on MapReduce was also studied in recent work [13, 56, 57]. However, these works are limited to co-clustering algorithms [56], subspace clustering [57], and metric k-centers and metric k-median with the assumption that all pairwise similarities are pre-computed and provided explicitly [13].

The approach proposed in this thesis aims at supporting all types of kernels, while being scalable and efficient when implemented on MapReduce. Our approach preserves all the advantages of kernel methods by being applicable to all data formats, not just data in vectorized forms. We keep our approach generic by defining a whole family of low-dimensional embedding methods characterized by only four properties, and we show that any embedding method that satisfies these four properties can be employed for achieving the goals of our approach.

[1] http://wiki.apache.org/cassandra/ArchitectureSSTable

Chapter 3

Scaling Kernel k-Means on MapReduce

This chapter describes the details of the proposed scalable kernel k-means approach, which is based on the observation that the linear k-means algorithm can be implemented in a quite efficient manner on MapReduce. Accordingly, reducing the non-linear kernel k-means to an appropriate linear variant of the k-means algorithm allows for scaling the kernel k-means algorithm on MapReduce. This reduction is possible if one can find a low-dimensional representation (embedding) for each input data instance, such that clustering the embeddings using the linear k-means algorithm provides close approximate clustering results to the clustering output of applying the kernel k-means algorithm to the original dataset.

Obviously, the most challenging part of the proposed approach is how to perform that reduction in a Mapreduce-efficient manner while ensuring that clustering accuracy is not degraded as a result of that reduction. We define a set of properties that ensures the efficiency and the correctness of the reduction/embedding step. Afterwards, we provide a unified strategy for parallelizing the computations of the embeddings and clustering them using the linear k-means algorithm. Further, we present three practical embedding methods that satisfy the defined properties. Finally, we study the computational complexities of all the proposed algorithms and analytically prove the scalability and the efficiency of our approach. Parts of the work presented in this chapter appear in our papers [58–60].

3.1 Approximate Nearest Centroid Embeddings

This section defines a family of embeddings, called Approximate Nearest Centroid (APNC) embeddings, that can be used to scale the kernel k-means on MapReduce. Essentially, we aim at

embeddings that: (1) can be computed in a MapReduce-efficient manner, and (2) can approximate the cluster assignment step of the kernel k-means on MapReduce (Eq. 2.7). We start with defining a set of properties which an embedding should have for the aforementioned conditions to be satisfied. In the following section, we show how these properties can be used to develop a MapReduce-efficient algorithm for kernel k-means.

Let i be a data instance, $\boldsymbol{\phi} = \Phi_{:i}$ be a vector corresponding to i in the kernel space implicitly defined by the kernel function. Let $f : \mathbb{R}^d \rightarrow \mathbb{R}^m$ be an embedding function that maps $\boldsymbol{\phi}$ to a target vector $\boldsymbol{y}$, i.e., $\boldsymbol{y} = f(\boldsymbol{\phi})$. In order to use $f(\boldsymbol{\phi})$ with the proposed MapReduce algorithm, the following properties have to be satisfied.

Property 3.1.1 *$f(\boldsymbol{\phi})$ is a linear map, i.e., $\boldsymbol{y} = f(\boldsymbol{\phi}) = T\boldsymbol{\phi}$, where $T \in \mathbb{R}^{m \times d}$.*

If this property is satisfied, then for any cluster c, the embedding of its centroid is the same as the centroid of the embeddings of the data instances that belong to that cluster:

$$\bar{\boldsymbol{y}}^{(c)} = f\left(\bar{\boldsymbol{\phi}}^{(c)}\right) = \frac{1}{n_c} \sum_{j \in \mathcal{P}_c} f\left(\boldsymbol{\phi}^{(j)}\right) = \frac{1}{n_c} \sum_{j \in \mathcal{P}_c} \boldsymbol{y}^{(j)} ,$$

where $\bar{\boldsymbol{y}}^{(c)}$ is the embedding of the centroid $\bar{\boldsymbol{\phi}}^{(c)}$.

Property 3.1.2 *$f(\boldsymbol{\phi})$ is kernelized.*

In order for this property to be satisfied, we restrict the columns of the transformation matrix T to be in the subspace of a subset of data instances $\mathcal{L} \subseteq \mathcal{D}$, $|\mathcal{L}| = l$ and $l \leq n$

$$T = R\Phi_{:\mathcal{L}}^T .$$

Substituting in $f(\boldsymbol{\phi})$ gives

$$\boldsymbol{y} = f(\boldsymbol{\phi}) = T\boldsymbol{\phi} = R\Phi_{:\mathcal{L}}^T\boldsymbol{\phi} = RK_{\mathcal{L}i} , \tag{3.1}$$

where $K_{\mathcal{L}i}$ is the kernel matrix between the set of instances $\mathcal{L}$ and the i-th data instance, and $R \in \mathbb{R}^{m \times l}$. We refer to R as the embedding coefficients matrix.

Suppose the set $\mathcal{L}$ definied in Property 3.1.2 consists of q disjoint subsets $\mathcal{L}^{(1)}$,$\mathcal{L}^{(2)}$,..., and $\mathcal{L}^{(q)}$.

Property 3.1.3 *The embedding coefficients matrix R is in a block-diagonal form:*

$$R = \begin{bmatrix} R^{(1)} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & R^{(q)} \end{bmatrix} ,$$

where q is the number of blocks and the b-th sub-matrix $R^{(b)}$ along with its corresponding subset of data instances $\mathcal{L}^{(b)}$ can be computed and fit in the memory of a single machine.

It should be noted that different embeddings of the defined family differ in their definitions of the coefficients matrix R.

Property 3.1.4 *There exists a function $e(\cdot, \cdot)$ that approximates the ℓ_2-distance between each data point i and the centroid of cluster c in terms of their embeddings $\boldsymbol{y}^{(i)}$ and $\bar{\boldsymbol{y}}^{(c)}$ only, i.e.,*

$$\exists \; e(\cdot, \cdot) : \; \left\| \boldsymbol{\phi}^{(i)} - \bar{\boldsymbol{\phi}}^{(c)} \right\|_2 \approx \beta \, e\left(\boldsymbol{y}^{(i)}, \, \bar{\boldsymbol{y}}^{(c)}\right) \;\; \forall \; i, c \;\; ,$$

where β is a constant.

This property allows for approximating the cluster assignment step of the kernel k-means defined in Eq.(2.7) as

$$\tilde{\pi}(i) = \arg\min_{c} \; e\left(\boldsymbol{y}^{(i)}, \, \bar{\boldsymbol{y}}^{(c)}\right) \; . \tag{3.2}$$

Since β is a constant, $\tilde{\pi}(i)$ will result in a cluster assignment for i close to $\pi(i)$ of the kernel k-means given by Eq.(2.7).

3.2 Efficient MapReduce-Based Parallelization Strategy

In this section, we show how the four properties of APNC embeddings can be exploited to develop an efficient and unified parallel MapReduce algorithm for kernel k-means. We start with the algorithm for computing the corresponding embedding for each data instance, then explain how to use these embeddings for approximating the kernel k-means.

Algorithm 2 APNC Embedding on MapReduce

Input: Distributed data points $\mathcal{D}$, Kernel function $\kappa(.,.)$, Embedding coefficients matrix R, Sample data points $\mathcal{L}$, Number of embedding blocks q
Output: Embedding matrix Y

1: **for** b = 1:q
2: ***map:***
3: Load $\mathcal{L}^{(b)}$ and $R^{(b)}$
4: **foreach** $< i, \mathcal{D}\{i\} >$
5: $K_{\mathcal{L}^{(b)}i} \leftarrow \kappa\left(\mathcal{L}^{(b)}, \mathcal{D}\{i\}\right)$
6: $\boldsymbol{y}^{(i)}_{[b]} \leftarrow R^{(b)} K_{\mathcal{L}^{(b)}i}$
7: emit(i,$\boldsymbol{y}^{(i)}_{[b]}$)
8: **end**
9: **end**
10: ***map:***
11: **foreach** $< i, \boldsymbol{y}^{(i)}_{[1]}, \boldsymbol{y}^{(i)}_{[2]}, ..., \boldsymbol{y}^{(i)}_{[q]} >$
12: $Y_{:i} \leftarrow \text{join}\left(\boldsymbol{y}^{(i)}_{[1]}, \boldsymbol{y}^{(i)}_{[2]}, ..., \boldsymbol{y}^{(i)}_{[q]}\right)$
13: emit(i,$Y_{:i}$)
14: **end**

3.2.1 APNC Embedding on MapReduce

From Property 3.1.2 and Property 3.1.3, the embedding $\boldsymbol{y}^{(i)}$ of a data instance i is given by

$$\boldsymbol{y}^{(i)} = \begin{bmatrix} R^{(1)} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & R^{(q)} \end{bmatrix} K_{\mathcal{L}i} \,, \tag{3.3}$$

for a set of selected data points $\mathcal{L}$. The set $\mathcal{L}$ consists of q disjoint subsets $\mathcal{L}^{(1)}$,$\mathcal{L}^{(2)}$,..., and $\mathcal{L}^{(q)}$. So, the vector $K_{\mathcal{L}i}$ can then be written in the form of q blocks as $K_{\mathcal{L}i} = [K^T_{\mathcal{L}^{(1)}i} K^T_{\mathcal{L}^{(2)}i} \dots K^T_{\mathcal{L}^{(q)}i}]^T$. Accordingly, the embedding formula in Eq. (3.3) can be written as

$$\boldsymbol{y}^{(i)} = \begin{bmatrix} R^{(1)} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & R^{(q)} \end{bmatrix} \begin{bmatrix} K_{\mathcal{L}^{(1)}i} \\ K_{\mathcal{L}^{(2)}i} \\ \vdots \\ K_{\mathcal{L}^{(q)}i} \end{bmatrix} = \begin{bmatrix} R^{(1)} K_{\mathcal{L}^{(1)}i} \\ R^{(2)} K_{\mathcal{L}^{(2)}i} \\ \vdots \\ R^{(q)} K_{\mathcal{L}^{(q)}i} \end{bmatrix} . \tag{3.4}$$

As per Property 3.1.3, each block $R^{(b)}$ and the sample instances $\mathcal{L}^{(b)}$ used to compute its corresponding $K_{\mathcal{L}^{(b)}i}$ are assumed to fit in the memory of a single machine. This suggests computing $\boldsymbol{y}^{(i)}$ in a piecewise fashion, where each portion $y^{(i)}_{[b]}$ is computed separately using its corresponding $R^{(b)}$ and $\mathcal{L}^{(b)}$.

As mentioned in Section 2.1, the input blocks are processed in parallel in the *map* phase, and each input block is processed sequentially. Our embedding algorithm on MapReduce computes the embedding portions of all data instances in rounds of q iterations. In each iteration, each *mapper* loads the corresponding coefficient block $R^{(b)}$ and data samples $\mathcal{L}^{(b)}$ in its memory. Afterwards, for each data point, the vector $K_{\mathcal{L}^{(b)}i}$ is computed using the provided kernel function and then used to compute the embedding portion as $\boldsymbol{y}^{(i)}_{[b]} = R^{(b)}K_{\mathcal{L}^{(b)}i}$. Finally, in a single *map* phase, the portions of each data instance i are concatenated together to form the embedding $\boldsymbol{y}^{(i)}$. It is important to note that the embedding portions of each data point will be stored on the same machine, which means that the concatenation phase has no network cost. The only network cost required by the whole embedding algorithm is only from loading the sub-matrices $R^{(b)}$ and $\mathcal{L}^{(b)}$ once for each b. Algorithm 2 outlines the embedding steps on MapReduce. We denote each key-value pair of the input dataset $\mathcal{D}$ as $< i, \mathcal{D}\{i\} >$ where i refers to the index of the data instance $\mathcal{D}\{i\}$.

3.2.2 APNC Clustering on MapReduce

To parallelize the clustering phase on MapReduce, we make use of Properties 3.1.1 and 3.1.4. As mentioned in Section 2.5.1, in each kernel k-means iteration, a data instance is assigned to its closet cluster centroid given by Eq. (2.8). Property 3.1.4 tells us that each data instance i can be approximately assigned to its closest cluster using only its embedding $\boldsymbol{y}^{(i)}$ and the embeddings of the current centroids. Further, Property 3.1.1 allows us to compute updated embeddings for cluster centroids, using the embeddings of the data instances assigned to each cluster.

Let $\bar{Y}$ be a matrix whose columns are the embeddings of the current centroids. Our MapReduce algorithm for the clustering phase parallelizes each kernel k-means iteration by loading the current centroids matrix $\bar{Y}$ to the memory of each *mapper*, and using it to assign a cluster to each data point represented by its embedding $\boldsymbol{y}^{(i)}$. Afterwards, the embeddings assigned to each cluster are grouped and averaged in a separate *reducer*, to find an updated matrix $\bar{Y}$ to be used in the following iteration. To minimize the network communication cost, we maintain an in-memory matrix Z whose columns are the summation of the embeddings of the data instances assigned to each cluster. We also maintain a vector $\boldsymbol{g}$ of the number of data instances in each cluster. We only move Z and $\boldsymbol{g}$ of each *mapper* across the network to the *reducers* that compute the updated

Algorithm 3 APNC Clustering on MapReduce

Input: Distributed embeddings matrix Y, Embedding dimensionality m, Number of clusters k, Discrepancy function $e(.,.)$
Output: Cluster centroids $\bar{Y}$

Generate initial k centroids $\bar{Y}$
repeat until convergence
  map:
    Load $\bar{Y}$
    Initialize $Z \leftarrow [0]_{m \times k}$ and $\mathbf{g} \leftarrow [0]_{k \times 1}$
    foreach $< i, Y_{:i} >$
      $\hat{c} = \arg\min_c e(Y_{:i}, \bar{Y}_{:c})$
      $Z_{:\hat{c}} \leftarrow Z_{:\hat{c}} + Y_{:i}$
      $\mathbf{g}_{\hat{c}} \leftarrow \mathbf{g}_{\hat{c}} + 1$
    end
    for c = 1:k
      emit(c,$< Z_{:c}, \mathbf{g}_c >$)
    end
  reduce:
    foreach $< c, \mathcal{Z}_c, \mathcal{G}_c >$
      $\bar{Y}_{:c} \leftarrow \left(\sum_{Z_{:c} \in \mathcal{Z}_c} Z_{:c}\right) / \left(\sum_{\mathbf{g}_c \in \mathcal{G}} \mathbf{g}_c\right)$
      emit(c,$\bar{Y}_{:c}$)
    end
end

$\bar{Y}$.[1] Algorithm 3 outlines the clustering steps on MapReduce.

3.3 APNC Embedding via Nyström Method

In this section, we develop our first instance of APNC embeddings based on the popular Nyström Method. In principle, one way to preserve the objective function of the cluster assignment step given by Equations (2.7) and (2.8) is to find a low-rank kernel matrix $\tilde{K}$ over the data instances, such that $K \approx \tilde{K}$. Using this kernel matrix in Eq. (2.8) results in a cluster assignment which is very close to the assignment obtained using the original kernel k-means algorithm.

[1] The in-memory Z and $\boldsymbol{g}$ can also be replaced by a MapReduce combiner [9]

Algorithm 4 APNC Coefficients via Nyström Method

Input: Distributed n data instances $\mathcal{D}$, Kernel function $\kappa(.,.)$, Number of samples l, Target dimensionality m.

Output: Sample data instances $\mathcal{L}$, Embedding coefficients matrix R.

map:
  for $< i, \mathcal{D}\{i\} >$
    with probability l/n, emit($0, \mathcal{D}\{i\}$)
  end
reduce:
  for $\mathcal{L} \leftarrow$ all values $\mathcal{D}\{i\}$
    $K_{\mathcal{L}\mathcal{L}} \leftarrow \kappa(\mathcal{L}, \mathcal{L})$
    $[\tilde{V}, \tilde{\Lambda}] \leftarrow$ eigen($K_{\mathcal{L}\mathcal{L}}, m$)
    $R \leftarrow \tilde{\Lambda}^{-1/2} \tilde{V}^T$
    emit($< S, R >$)
  end

If the low-rank approximation $\tilde{K}$ can be decomposed into $W^T W$ where $W \in \mathbb{R}^{m \times n}$ and $m \ll n$, then the columns of W can be directly used as an embedding that approximates the l_2 distance between data instance i and the centroid of cluster c as

$$\left\| \boldsymbol{\phi}^{(i)} - \bar{\boldsymbol{\phi}}^{(c)} \right\|_2 \approx \left\| \boldsymbol{w}^{(i)} - \bar{\boldsymbol{w}}^{(c)} \right\|_2 . \tag{3.5}$$

To prove that, the right-hand side can be simplified to

$$\begin{aligned} &\boldsymbol{w}^{(i)T} \boldsymbol{w}^{(i)} - 2\boldsymbol{w}^{(i)T} \bar{\boldsymbol{w}}^{(c)} + \bar{\boldsymbol{w}}^{(c)T} \bar{\boldsymbol{w}}^{(c)} \\ &= \tilde{K}_{ii} - \frac{2}{n_c} \sum_{a \in \mathcal{P}_c} \tilde{K}_{ia} + \frac{1}{n_c^2} \sum_{a,b \in \mathcal{P}_c} \tilde{K}_{ab} \end{aligned}$$

The right-hand side is an approximation of the objective function of Eq. (2.8).

There are many low-rank decompositions that can be calculated for the kernel matrix K, including the very accurate eigenvalue decompositions. However, the low-rank approximation used has to satisfy the properties defined in Section 3.1, and accordingly can be implemented on MapReduce in an efficient manner.

One well-known method for the low-rank approximation of kernel matrices is the Nyström approximation [32]. Recall from Section 2.4.1 the the Nyström approximation is computed as

$$\tilde{K} = D^T A^{-1} D \, , \tag{3.6}$$

where $|\mathcal{L}| = l \ll n$, $A \in \mathbb{R}^{l \times l}$ is the kernel matrix over the data instances in $\mathcal{L}$, and $D \in \mathbb{R}^{l \times n}$ is the kernel matrix between the data instances in $\mathcal{L}$ and all data instances. In order to obtain a low-rank decomposition of $\tilde{K}$, the Nyström method calculates the eigendecomposition of the small matrix A as $A \approx U \Lambda U^T$, where $U \in \mathbb{R}^{l \times m}$ is the matrix whose columns are the leading-m eigenvectors of A, and $\Lambda \in \mathbb{R}^{m \times m}$ is the matrix whose diagonal elements are the leading m eigenvalues of A. Substituting this in Eq. (3.6) results in

$$\tilde{K} = D^T U \Lambda^{-1} U^T D \,. \tag{3.7}$$

This means that a low-rank decomposition can be obtained as $\tilde{K} = W^T W$ where $W = \Lambda^{-1/2} U^T D$. It should be noted that this embedding satisfies Properties 3.1.1 and 3.1.2 as $D = \Phi_{:\mathcal{L}}^T \Phi$, and accordingly $\boldsymbol{y}^{(i)} = W_{:i} = \Lambda^{-1/2} U^T \Phi_{:\mathcal{L}}^T \boldsymbol{\phi}^{(i)}$. Further, Equation (3.5) tells us that $e\left(\boldsymbol{y}^{(i)}, \bar{\boldsymbol{y}}^{(c)}\right) = \left\| \boldsymbol{y}^{(i)} - \bar{\boldsymbol{y}}^{(\boldsymbol{c})} \right\|_2$ can be used to approximate the ℓ_2-distance in Eq. (2.7), which satisfies Property 3.1.4 of the APNC family.

The embedding coefficient matrix $R = \Lambda^{-1/2} U^T$ is a special case of that described in Property 3.1.3, which consists of one block of size $m \times l$, where l is the number of instances used to calculate the Nyström approximation, and m is the rank of the eigen-decomposition used to compute both Λ and U. It can be assumed that R is computed and fits in the memory of a single machine, since an accurate Nyström approximation can usually be obtained using a very few samples and $m \leq l$. Algorithm 4 outlines the MapReduce algorithm of computing the coefficients matrix R. The algorithm uses the *map* phase to iterate over the input dataset in parallel, to uniformally sample l data instances. The sampled instances are then moved to a single *reducer* that computes R as described above. $\tilde{V}$ and $\tilde{\Lambda}$ denote the eigenvectors and eigenvalues matrices computed using the truncated eigen-decomposition function *eigen($K_{\mathcal{LL}}$,m)* in line 8 of the algorithm.

3.4 APNC Embedding via Ensemble Nyström Method

The Nsyström embedding can be extended by the use of the ensemble Nyström method [37] outlined in section 2.4.1. A kernel low-rank approximation is computed in the ensemble Nyström method as a weighted summation of q Nyström approximations as

$$\tilde{K} = \mu_1 {D^{(1)}}^T {A^{(1)}}^{-1} D^{(1)} + \mu_1 {D^{(2)}}^T {A^{(2)}}^{-1} D^{(2)} + \ldots + \mu_1 {D^{(q)}}^T {A^{(q)}}^{-1} D^{(q)} \,, \tag{3.8}$$

which can be written in a matrix form as

$$\tilde{K} = D^T \begin{bmatrix} \mu_1 A^{(1)^{-1}} & 0 & \dots & 0 \\ 0 & \mu_2 A^{(2)^{-1}} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \mu_q A^{(q)^{-1}} \end{bmatrix} D, \quad D = \begin{bmatrix} D^{(1)} \\ D^{(2)} \\ \vdots \\ D^{(q)} \end{bmatrix} \tag{3.9}$$

Equivalently,

$$\tilde{K} = D^T \begin{bmatrix} \frac{1}{\mu_1} A^{(1)} & 0 & \dots & 0 \\ 0 & \frac{1}{\mu_2} A^{(2)} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{1}{\mu_q} A^{(q)} \end{bmatrix}^{-1} D \tag{3.10}$$

Accordingly, Eq. 3.10 can be rewritten in the form of $\tilde{K} = W^T W$ where W is given by

$$W = \begin{bmatrix} \frac{1}{\mu_1} A^{(1)} & 0 & \dots & 0 \\ 0 & \frac{1}{\mu_2} A^{(2)} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{1}{\mu_q} A^{(q)} \end{bmatrix}^{-1/2} D \tag{3.11}$$

Equivalently,

$$W = \begin{bmatrix} \frac{1}{\sqrt{\mu_1}} A^{(1)^{-1/2}} & 0 & \dots & 0 \\ 0 & \frac{1}{\sqrt{\mu_2}} A^{(2)^{-1/2}} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{1}{\sqrt{\mu_q}} A^{(q)^{-1/2}} \end{bmatrix} D \tag{3.12}$$

Suppose each positive semi-definite block $A^{(b)}$ of Eq. 3.12 is decomposed in the form

$$A^{(b)} = U^{(b)} \Lambda^{(b)} U^{(b)^T},$$

where the columns of $U^{(b)}$ are the $m^{(b)}$ leading eigenvectors of $A^{(b)}$, and $\Lambda^{(b)}$ is a diagonal matrix whose diagonal entries are the $m^{(b)}$ leading eigenvalues of $A^{(b)}$. Then, each $A^{(b)^{(-1/2)}}$ in Eq. 3.12 can be computed as

$$A^{(b)^{-1/2}} = \Lambda^{(b)^{-1/2}} U^{(b)^T}.$$

As shown in Section 3.3, decomposing the kernel matrix in the form $\tilde{K} = W^T W$ provides an embedding method that can approximate the ℓ_2-distance of the kernel k-means. Equation 3.12 tells us that the embedding function provided by the ensemble Nyström method is in the form

$$\boldsymbol{y}^{(i)} = f\left(\boldsymbol{\phi}^{(i)}\right) = W_{:i} = \begin{bmatrix} \frac{1}{\sqrt{\mu_1}} A^{(1)^{-1/2}} & 0 & \dots & 0 \\ 0 & \frac{1}{\sqrt{\mu_2}} A^{(2)^{-1/2}} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{1}{\sqrt{\mu_q}} A^{(q)^{-1/2}} \end{bmatrix} D_{:i} \tag{3.13}$$

where $D_{:i}$ is an l-dimensional vector of the kernel between the data instance i and all the sample instances in $\mathcal{L}^{(1)}$, $\mathcal{L}^{(2)}$, ..., and $\mathcal{L}^{(q)}$ and $l = \sum_{b=1}^{q} |\mathcal{L}^b|$, i.e.

$$D_{:i} = \begin{bmatrix} \Phi_{:\mathcal{L}^{(1)}} & \Phi_{:\mathcal{L}^{(2)}} & \dots & \Phi_{:\mathcal{L}^{(q)}} \end{bmatrix}^T \boldsymbol{\phi}^{(i)}$$

The definition of $D_{:i}$ above shows that the ensemble Nyström embedding given by Eq. 3.13 satisfies properties 3.1.1 and 3.1.2 of APNC embeddings. Further, Eq. 3.13 shows that the embedding coefficients matrix R of the ensemble Nyström embedding is in a block-diagonal form, where

$$R = \begin{bmatrix} \frac{1}{\sqrt{\mu_1}} A^{(1)^{-1/2}} & 0 & \dots & 0 \\ 0 & \frac{1}{\sqrt{\mu_2}} A^{(2)^{-1/2}} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{1}{\sqrt{\mu_q}} A^{(q)^{-1/2}} \end{bmatrix} \tag{3.14}$$

Thus, the embedding given by Eq. 3.13 satisfies property 3.1.3 of APNC embeddings. Similar to the Nyström embedding described in Section 3.3, the ℓ_2-distance of the kernel k-means can be approximated directly by the ℓ_2-distance of the ensemble Nyström embeddings as

$$\left\| \boldsymbol{\phi}^{(i)} - \bar{\boldsymbol{\phi}}^{(c)} \right\|_2 \approx \left\| \boldsymbol{y}^{(i)} - \bar{\boldsymbol{y}}^{(c)} \right\|_2 , \tag{3.15}$$

Algorithm 5 APNC Coefficients via Ensemble Nyström Method

Input: Distributed n data instances $\mathcal{D}$, Kernel function $\kappa(.,.)$, Number of samples l, Target dimensionality m, Ensemble size q

Output: Sample data instances $\mathcal{L}$, Embedding coefficients matrix R.

1: ***map:***
2: **for** $< i, \mathcal{D}\{i\} >$
3: Draw r uniformally from 1 to n
4: **if** $r \leq l$
5: emit($\lceil rq/l \rceil$,$\mathcal{D}\{i\}$)
6: **end**
7: **end**
8: ***reduce:***
9: **foreach** (b,$\mathcal{L}^{(b)}$)
10: $A \leftarrow \kappa\left(\mathcal{L}^{(b)}, \mathcal{L}^{(b)}\right)$
11: $[\tilde{V}, \tilde{\Lambda}] \leftarrow$ eigen(A,$\frac{m}{q}$)
12: $R^{(b)} \leftarrow \tilde{\Lambda}^{-1/2}\tilde{V}^T$
13: emit($< b, (\mathcal{L}^{(b)}, R^{(b)}) >$)
14: **end**

where $\boldsymbol{y}^{(i)}$ is given by Eq. 3.13 and $\bar{\boldsymbol{y}}^{(c)}$ is the centroid of the embeddings assigned to the cluster c. Accordingly, the ensemble Nyström embedding satisfies property 3.1.4 of APNC embeddings.

Algorithm 5 outlines the MapReduce algorithm of computing the coefficients matrix R given by Eq. 3.14. For simplicity, we assume that blocks of R are computed with the same number of samples. Similar to Algorithm 4 above, in the *map* phase the algorithm iterates over the input dataset in parallel to uniformally sample l data instances. The sampled instances are then moved to one of q possible *reducers*, each of which computes a block of R in parallel as described above.

3.5 APNC Embedding via Stable Distributions

In this section, we develop our third embedding method based on the results of Indyk [61], which showed that the ℓ_p-norm of a d-dimensional vector $\boldsymbol{v}$ can be estimated by means of p-stable distributions. Given a d-dimensional vector $\boldsymbol{r}$ whose entries are i.i.d. samples drawn from

a p-stable distribution over $\mathbb{R}$, the ℓ_p-norm of $\boldsymbol{v}$ is given by

$$||\boldsymbol{v}||_p = \alpha \mathbb{E}[|\sum_{i=1}^{d} \boldsymbol{v}_i \boldsymbol{r}_i|] \,, \tag{3.16}$$

for some positive constant α. It is known that the standard Gaussian distribution $\mathcal{N}(0,1)$ is 2-stable [61], which means that it can be employed to compute the ℓ_2-norm of Eq. (2.8) as

$$\left\|\boldsymbol{\phi} - \bar{\boldsymbol{\phi}}\right\|_2 = \alpha \mathbb{E}[|\sum_{i=1}^{d} (\boldsymbol{\phi}_i - \bar{\boldsymbol{\phi}}_i) \boldsymbol{r}_i|] \,, \tag{3.17}$$

where d is the dimensionality of the space endowed by the used kernel function and the entries $\boldsymbol{r}_i \sim \mathcal{N}(0,1)$. The expectation above can be approximated by the sample mean of multiple values for the term $|\sum_{i=1}^{d} (\boldsymbol{\phi}_i - \bar{\boldsymbol{\phi}}_i) r_i|$ computed using m different vectors $\boldsymbol{r}$ each of which is denoted as $\boldsymbol{r}^{(j)}$. Thus, the ℓ_2-norm in Eq. 3.17 can be approximated as

$$\left\|\boldsymbol{\phi} - \bar{\boldsymbol{\phi}}\right\|_2 \approx \frac{\alpha}{m} \sum_{j=1}^{m} |\sum_{i=1}^{d} \left(\boldsymbol{\phi}_i \boldsymbol{r}_i^{(j)} - \bar{\boldsymbol{\phi}}_i \boldsymbol{r}_i^{(j)} \right)| \tag{3.18}$$

Define two m-dimensional embeddings $\boldsymbol{y}$ and $\bar{\boldsymbol{y}}$ such that $\boldsymbol{y}_j = \sum_{i=1}^{d} \boldsymbol{\phi}_i \boldsymbol{r}_i^{(j)}$ and $\bar{\boldsymbol{y}}_j = \sum_{i=1}^{d} \bar{\boldsymbol{\phi}}_i \boldsymbol{r}_i^{(j)}$ or equivalently, $\boldsymbol{y}_j = \boldsymbol{\phi}^T \boldsymbol{r}^{(j)}$ and $\bar{\boldsymbol{y}}_j = \bar{\boldsymbol{\phi}}^T \boldsymbol{r}^{(j)}$. Equation (3.18) can be expressed in terms of $\boldsymbol{y}$ and $\bar{\boldsymbol{y}}$ as

$$\left\|\boldsymbol{\phi} - \bar{\boldsymbol{\phi}}\right\|_2 \approx \frac{\alpha}{m} \sum_{j=1}^{m} |\boldsymbol{y}_j - \bar{\boldsymbol{y}}_j| = \frac{\alpha}{m} \left\|\boldsymbol{y} - \bar{\boldsymbol{y}}\right\|_1 \,. \tag{3.19}$$

Since all of $\boldsymbol{\phi}$, $\bar{\boldsymbol{\phi}}$ and $\boldsymbol{r}^{(j)}$ are intractable to explicitly work with, our next step is to kernelize the computations of $\boldsymbol{y}$ and $\bar{\boldsymbol{y}}$. Without loss of generality, let $\mathcal{T}_j = \{\hat{\boldsymbol{\phi}}^{(1)}, \hat{\boldsymbol{\phi}}^{(2)}, ..., \hat{\boldsymbol{\phi}}^{(t)}\}$ be a set of t randomly chosen data instances embedded and centered into the kernel space (i.e. $\hat{\boldsymbol{\phi}}^{(i)} = \boldsymbol{\phi}^{(i)} - \frac{1}{t} \sum_{j=1}^{t} \boldsymbol{\phi}^{(j)}$). According to the central limit theorem, the vector $\boldsymbol{r}^{(j)} = \frac{1}{\sqrt{t}} \sum_{\boldsymbol{\phi} \in \mathcal{T}_j} \boldsymbol{\phi}$ approximately follows a multivariate Gaussian distribution $\mathcal{N}(0, \Sigma)$, where Σ is the covariance matrix of the underlying distribution of all data instances embedded into the kernel space [28]. But according to our definition of $\boldsymbol{y}$ and $\bar{\boldsymbol{y}}$, the individual entries of $\boldsymbol{r}^{(j)}$ have to be independent and identically Gaussians. To fulfil that requirement, we make use of the fact that decorrelating the variables of a joint Gaussian distribution is enough to ensure that the individual variables are independent and marginally Gaussian. Using the whitening transform, $\boldsymbol{r}^{(j)}$ is then redefined as

$$\boldsymbol{r}^{(j)} = \frac{1}{\sqrt{t}} \tilde{\Sigma}^{-1/2} \sum_{\boldsymbol{\phi} \in \mathcal{T}^{(j)}} \boldsymbol{\phi} \,, \tag{3.20}$$

Algorithm 6 Approximate Kernel k-Means Using Stable Distributions

Input: Dataset $\mathcal{X}$ of n data instances, Kernel Function $\kappa(.,.)$,
APNC Parameters l, m, and t, Number of Clusters k
Output: Clustering Labels $\boldsymbol{l}$

$\mathcal{L} \leftarrow$ uniform sample of l data instances from $\mathcal{X}$
$K_{\mathcal{L}\mathcal{L}} \leftarrow \kappa(\mathcal{L}, \mathcal{L})$
$H \leftarrow I - \frac{1}{l} ee^T$
$K_{\mathcal{L}\mathcal{L}} \leftarrow HK_{\mathcal{L}\mathcal{L}}H$
$[V, \Lambda] \leftarrow eigen(K_{\mathcal{L}\mathcal{L}})$
$E \leftarrow \Lambda^{-1/2}V^T$
Initialize $R \leftarrow [0]_{m \times l}$
for $r = 1 : m$
    $\mathcal{T} \leftarrow$ select t unique values from 1 to p
    $R_{r:} = \sum_{v \in \mathcal{T}} E_{v:}$
end
$K_{:\mathcal{L}} \leftarrow \kappa(\mathcal{X}, \mathcal{L})$
$Y \leftarrow RK_{:\mathcal{L}}^T$
$\boldsymbol{l} \leftarrow k$-Means($Y$, k, ℓ_1) *// Lloyd k-means with the ℓ_1-distance.*

where $\tilde{\Sigma}$ is an approximate covariance matrix estimated using a sample of l data points embedded into the kernel space and centered as well. We denote the set of the l data points as $\mathcal{L}$.

With $\boldsymbol{r}^{(j)}$ defined as in Eq. (3.20), the computation of $\boldsymbol{y}$ and $\bar{\boldsymbol{y}}$ can be fully kernelized by following similar simplification steps to those in [28]. Accordingly, $\boldsymbol{y}$ and $\bar{\boldsymbol{y}}$ can be computed as follows: let $K_{\mathcal{L}\mathcal{L}}$ be the kernel matrix of $\mathcal{L}$, and define a centering matrix $H = I - \frac{1}{l}\boldsymbol{e}\boldsymbol{e}^T$ where I is an $l \times l$ identity matrix, and $\boldsymbol{e}$ is a vector of all ones. Denote the inverse square root of the centered version of $K_{\mathcal{L}\mathcal{L}}$ as E.[2] The embedding of a vector $\boldsymbol{\phi}$ is then given by:

$$\boldsymbol{y} = f(\boldsymbol{\phi}) = R\Phi_{:\mathcal{L}}^T\boldsymbol{\phi} \tag{3.21}$$

such that for $j = 1$ to m, $R_{j:} = \boldsymbol{s}^T EH$ where $\boldsymbol{s}$ is an l-dimensional binary vector indexing t randomly chosen values from 1 to l for each j. Algorithm 6 outlines the steps of the approximate kernel k-means algorithm using stable distributions.

Now, we show that the embedding function f defined in Eq. (3.21) is an APNC Embedding function. It is clear from Eq. (3.21) that f is a linear map in a kernelized form which satisfies

[2]The centered version of $K_{\mathcal{L}\mathcal{L}}$ is given by $HK_{\mathcal{L}\mathcal{L}}H$. Its inverse square root can be computed as $\Lambda^{-1/2}V^T$ where Λ is a diagonal matrix of the eigenvalues of $HK_{\mathcal{L}\mathcal{L}}H$, and V is the eigenvector matrix of $HK_{\mathcal{L}\mathcal{L}}H$.

Algorithm 7 APNC Coefficients via Stable Distributions

Input: Distributed n data instances $\mathcal{D}$, Kernel function $\kappa(.,.)$, Number of samples l, Target dimensionality m, Tuning parameter t.
Output: Sample data instances $\mathcal{L}$, Embedding coefficients matrix R.

map:
  for $< i, \mathcal{D}\{i\} >$
    with probability l/n, emit($0, \mathcal{D}\{i\}$)
  end
reduce:
  for $\mathcal{L} \leftarrow$ all values $\mathcal{D}\{i\}$
    $K_{\mathcal{L}\mathcal{L}} \leftarrow \kappa(\mathcal{L}, \mathcal{L})$
    $H \leftarrow I - \frac{1}{l} e e^T$
    $[V, \Lambda] \leftarrow$ eigen$(H K_{\mathcal{L}\mathcal{L}} H)$
    $E \leftarrow \Lambda^{-1/2} V^T$
    for r = 1:m
      $\mathcal{T} \leftarrow$ select t unique values from 1 to l
      $R_{r:} = \sum_{v \in \mathcal{T}} E_{v:}$
    end
    emit($< S, R >$)
  end

Properties 3.1.1 and 3.1.2. Equation (3.19) shows that the ℓ_2-norm of the difference between a data point $\boldsymbol{\phi}$ and a cluster centroid $\bar{\boldsymbol{\phi}}$ can be approximated up to a constant by $e(\boldsymbol{y}, \bar{\boldsymbol{y}}) = \|\boldsymbol{y} - \bar{\boldsymbol{y}}\|_1$ which satisfies Property 3.1.4 of APNC family. The coefficients matrix R in Eq. (3.21) is of a single block, which can be assumed to be computable in the memory of a single commodity machine. That assumption is justified by observing that R is computed using a sample of a few data instances that are used to conceptually estimate the covariance matrix of the data distribution. Furthermore, the target dimensionality, denoted as m in Eq. (3.21), determines the sample size used to estimate the expectation in Eq. (3.17), which also can be estimated by a small number of samples. We validate the assumptions about l and m in our experiments. This accordingly satisfies Property 3.1.3. We outline the MapReduce algorithm for computing the coefficients matrix R, defined by Eq. (3.21) in Algorithm 7. Similar to Algorithm 4, we sample l data instance in the *map* phase, and then R is computed using the sampled data instances in a single *reducer*.

3.6 Analysis

In this section, the proposed algorithms are analyzed, in terms of their memory requirements per node and their time and network communication complexities. For clarity, the network communication complexity is analyzed separate from the time complexity, and the reported time complexities indicate only the processing times of the *map* and *reduce* functions of each algorithm. Let n be the total number of data instances being clustered. In Mapreduce settings, input data are partitioned into b physical blocks distributed across different nodes. Input blocks are processed in parallel by a number of worker threads in each *map* phase. It could happen that the number of input blocks might be greater than the number of worker threads. In that case, one or more worker threads will process more than one input block in each pass over the input dataset in the *map* phase. We use p to denote the maximum number of input blocks processed by a single *mapper* in each pass over the input dataset. If the number of worker threads is greater than or equal to the number of the input blocks, the value of p will be 1.

In this section, let $\hat{l}$ denote the cardinality of the largest subset of data instances used to compute the embeddings as defined in Section 3.1 (i.e. $\hat{l} = \arg\max_i |\mathcal{L}^{(i)}|$). Also, let the size of each block of R be bounded by $\hat{m} \times \hat{l}$. So, $\hat{m} = m$ in Algorithms 4 and 7 while $\hat{m} = \frac{m}{q}$ in Algorithm 5. Similar to Section 3.2 above, k is used to denote the target number of clusters, l is the total number of data instances used to compute the corresponding APNC coefficients matrix R, q is the number of blocks of R, and m denotes the target dimensionality of the computed embeddings. In addition, let s denote the size of each input data instance.

As outlined in Algorithm 2, the space complexity per node to compute the embeddings is determined by the amount of memory required to store each block of R and its corresponding subset of data instances (loaded in line:3). Thus, the space complexity per node of the embedding step is $\mathcal{O}(\hat{l}(\hat{m} + s))$. The only network communication needed in the embedding step is that of loading each block of R and its corresponding subset of data instances by each *mapper* from the distributed file system. Hence, the network cost of the embedding step is $\mathcal{O}(bl(\hat{m} + s))$[3]. The time complexity is dominated by the matrix-vector multiplication in line:6 which is of $\mathcal{O}(\hat{m}\hat{l})$ for each input data instance. The number of data instances per each physical input block is $\frac{n}{b}$ and accordingly, the time complexity of the embedding step is $\mathcal{O}(\frac{n}{b}pm\hat{l})$.

It can be noticed in Algorithm 3 that the required memory per node in the clustering step is that needed to store the current cluster centroids (loaded in line:4). Thus, the space complexity of the clustering step is $\mathcal{O}(mk)$. The network communication per iteration is incurred by two sources. The first is from loading the cluster centroids from the distributed file system in line:4, and the second is from sending the new cluster assignment information to the *reducer* in line:12.

[3]Note that $q\hat{l}$ was replaced with l

The total network communication cost per iteration from both sources is $\mathcal{O}(mkb)$. Suppose the required number of iterations for convergence is t. Then, the overall network cost of the clustering step is $\mathcal{O}(tbmk)$. The time complexity of computing the discrepancy between two m-dimensional embeddings using the function $e(.,.)$ is typically $\mathcal{O}(m)$. The time complexity of the clustering step is dominated by finding the nearest cluster centroid to each data point in each iteration (line:7) which is $\mathcal{O}(mk)$. Therefore, the overall times complexity of the clustering step is $\mathcal{O}(\frac{n}{b}ptmk)$.

The three embedding methods outlined in Algorithms 4, 5, and 7 have the same space, time, and network communication complexities. The required space complexity is $\mathcal{O}(\hat{l}^2)$, which is the memory required to store the pairwise kernel matrix of the data instances used to compute APNC coefficient matrix R in each algorithm. The time complexity is dominated by the eigen-decomposition step in each algorithm. The eigen-decomposition step is typically $\mathcal{O}(\hat{l}^3)$. However, in situations where the desired rank of the decomposition $\hat{m}$ is much less than $\hat{l}$ (i.e. $\hat{m} \ll \hat{l}$), the efficient yet accurate stochastic singular value decomposition (SSVD) [39] can be used to compute an approximate eigen-decomposition in Algorithms 4 and 5. Using the SSVD algorithm reduces the $\mathcal{O}(\hat{l}^3)$ time complexity of the decomposition step to $\mathcal{O}(\hat{l}^2\hat{m} + \hat{m}^3)$ [38], which will still be the most expensive step in Algorithms 4 and 5. The network communication cost is incurred only by moving the sampled data instances to the *reduce* function. Thus, the total network communication cost of the three algorithms is $\mathcal{O}(ls)$.

Table 3.1 summarizes the complexities of the proposed algorithms. It can be observed that the space complexities of all algorithms are independent from the dataset size n, which ensures the scalability of the proposed approach regardless of the input dataset size. Furthermore, in Section 4.2 it is empirically shown that quite reasonable clustering accuracy can be achieved using very small values for l and m. In addition, the table shows that the network communication costs of Algorithm 2 and Algorithm 3 are coarse-grained to be functions of the number of physical input blocks b, rather than of the number of input data instances n, which contributes significantly to the efficiency of the proposed approach. The network communication and time complexities of the three embedding proposed algorithms are independent from both n and b. The time complexities of the embedding and the clustering steps are both linear in n. However, increasing the number of data instances n translates into an increase in the number of data instances per block $\frac{n}{b}$, or an increase in the number of blocks b. In Mapreduce settings, the number of data instances per block is typically kept constant by setting a maximum value for the block size. On the other hand, increasing the number of blocks results in an increase to the number of blocks processed by each worker thread in order to do one pass over the entire dataset (i.e. the value of p is increased). However, the value of p can also be kept constant as n increases by adding more nodes/workers to the cluster infrastructure, which allows for restricting the increase in the running time of the clustering process to only the additional network communication cost incurred by the increase in

Algorithm 8 Deterministic Subset Sampling

Input: Distributed n data instances $\mathcal{D}$, Number of samples l
Output: Sample data instances $\mathcal{L}$

1: *[Method 1]* $S \leftarrow$ Draw l indexes from 1 to n
2: *[Method 2]* Load S from DFS
3: ***map:***
4: **for** $< i, \mathcal{D}\{i\} >$
5: **if** $i \in S$
6: emit($\mathcal{D}\{i\}$)
7: **endif**
8: **end**

the number of blocks.

Table 3.1: The space, time, and network communication complexities of the steps of the proposed approach.

Step	Space Complexity	Network Complexity	Time Complexity
Embedding (Alg. 2)	$\mathcal{O}(\hat{l}(\hat{m}+s))$	$\mathcal{O}(bl(\hat{m}+s))$	$\mathcal{O}(\frac{n}{b}pm\hat{l})$
Clustering (Alg. 3)	$\mathcal{O}(mk)$	$\mathcal{O}(tbmk)$	$\mathcal{O}(\frac{n}{b}ptmk)$
Nys-Coeff. (Alg. 4)	$\mathcal{O}(l^2)$	$\mathcal{O}(ls)$	$\mathcal{O}(\min(l^3, l^2m+m^3))$
Ens-Nys-Coeff. (Alg. 5)	$\mathcal{O}(\hat{l}^2)$	$\mathcal{O}(ls)$	$\mathcal{O}(\min(\hat{l}^3, \hat{l}^2m+m^3))$
SD-Coeff. (Alg. 7)	$\mathcal{O}(l^2)$	$\mathcal{O}(ls)$	$\mathcal{O}(l^3)$

3.7 Implementation Details

In this section, we discuss some implementation details that contribute to the efficiency and the scalability of the proposed kernel clustering approach.

3.7.1 Deterministic versus Probabilistic Sampling

It can be noticed from Algorithms 4, 5, and 7 that we used a probabilistic sampling approach to uniformly choose the subset of data instances that are used to compute the embeddings. There

Algorithm 9 Handling Empty Clusters in APNC Clustering

Input: Partial summations of the embeddings assigned to each cluster $\mathcal{Z}_c$, Number of embeddings assigned to each cluster $\mathcal{G}_c$, Backup embeddings $\mathcal{U}$

Output: Sample data instances $\mathcal{L}$

reduce:
  Load the backup embeddings $\mathcal{U}$
    foreach $< c, \mathcal{Z}_c, \mathcal{G}_c >$
      $g \leftarrow \sum_{\mathbf{g}_c \in \mathcal{G}_c} \mathbf{g}_c$
      if $g = 0$
        $\boldsymbol{u} \leftarrow$ remove a sample from $\mathcal{U}$
        $\bar{Y}_{:c} \leftarrow \boldsymbol{u}$
      else
        $\bar{Y}_{:c} \leftarrow \left(\sum_{Z_{:c} \in \mathcal{Z}_c} Z_{:c}\right) / g$
      endif
      emit(c,$\bar{Y}_{:c}$)
    end
end

are two alternative deterministic ways for sampling l data instances from n distributed instances. The first is to uniformly choose l distinct random indexes from 1 to n in a centralized way (on a single machine), then write the chosen indexes to the distributed file system before they are loaded by the *mappers* in the aforementioned algorithms. A data instance is then outputted to the *reducers* if its key (index) is in the loaded list of indexes. The second way is to rely on a pseudorandom number generator with a shared seed among all nodes. Each node first samples l unique indexes from 1 to n, and keeps the sampled indexes in its memory. An instance is outputted by a *mapper* if its index is in the list of chosen indexes. Algorithm 8 outlines the steps of the deterministic sampling approach. Clearly, the probabilistic sampling approach adopted in Algorithms 4, 5, and 7 is more efficient, and obviates the need to sample and store the set of indexes S in the memory of each node.

3.7.2 Handling Empty Clusters

One common issue with the k-means clustering algorithm is how to prevent empty clusters. In the centralized settings, the empty clusters problem is typically handled by assigning one particular (usually arbitrary) sample to the empty cluster after each iteration. However, in our

distributed APNC clustering algorithm (Algorithm 3), empty clusters can be detected only in the *reduce* phase, in which data instances are no longer accessible. Our simple and scalable approach to handing the empty clusters is based on choosing a subset of *backup* embeddings $\mathcal{U}$ in an initialization step of the algorithm. These backup embeddings are written to the distributed file system and loaded to the memory of the *reducer* of the APNC clustering algorithm in each iteration. Empty clusters are detected easily by the *reducer* after summing the total number of data instances assigned to each cluster. The *reducer* then chooses an arbitrary data instance out of the backup embeddings, and assigns it to the detected empty cluster. The number of backup embeddings $|\mathcal{U}|$ is a tuning parameter to the APNC Clustering algorithm. It should be at least equal to $k-1$, where k is the number of clusters. However, we suggest to set the size of $\mathcal{U}$ to a greater value than $k-1$, to avoid producing the same cluster assignments in the following iteration. The probabilistic sampling approach discussed in Section 3.7.1 can be employed to efficiently generate the backup embeddings file. Algorithm 9 shows a revised outline for the *reducer* of the APNC clustering algorithm with the proper steps for handling the empty clusters problem.

3.7.3 Convergence and Local Optima

The k-means algorithm is known to achieve a local optimal clustering solution. In one-node settings, one practical approach to tackling this problem is to run the k-means algorithm multiple times with different initial cluster assignments, and take the clustering result that achieves the best objective function (e.g. minimum least square error as defined in Eq. 2.6). A similar approach can be integrated in the proposed APNC clustering algorithm. In fact, one does not need to restart the clustering algorithm multiple times. Different runs with different initial conditions can be executed in parallel. Recall that in Algorithm 3 the current centroids matrix $\bar{Y}$ is loaded to the memory of all *mappers* to decide the closest centroid to each data point. One can obtain q different clustering solutions by using q matrices of clusters centroids, and letting each *mapper* assign each data instance to the closest centroid in each matrix $\bar{Y}^{(b)}$ for $b = 1, 2, ..., q$.

The stopping ("convergence") criterion used in our implementation is a fixed number of iterations. We believe this is the most efficient approach to terminating the k-means iterations in MapReduce settings. This criteria can be augmented by comparing the current centroids of each iterations to the previous centroids, to make sure that further iterations can still improve the clustering results.

Algorithm 10 Generating the Final Cluster Assignments

Input: Distributed n data instances $\mathcal{D}$, Final centroids $\bar{Y}$, Discrepancy function $e(.,.)$
Output: Cluster Assignments

1: Load $\bar{Y}$
2: ***map:***
3: **for** $< i, Y_{:i} >$
4: $\hat{c} = \arg\min_c e(Y_{:i}, \bar{Y}_{:c})$
5: emit(i,$\hat{c}$)
6: **end**

3.7.4 Clustering Output

In the APNC clustering algorithm (Algorithm 3), the last step of each iteration is to generate new clustering centroids. At the end of the last iteration, the final clustering centroids are considered as the output of the algorithm. In most applications, finding the cluster assignment of each data instance is needed. In that case, a *map*-only job should be used to assign each data instance to its closest final centroid. Algorithm 10 outlines the steps for generating the final cluster assignments.

Chapter 4

Experiments and Results

In this chapter, we present the experiments conducted to assess the effectiveness and the performance of the proposed algorithms. We also study the effect of each algorithm's tuning parameters on both the effectiveness and the efficiency of the proposed approach. In addition, we empirically demonstrate the scalability of the overall approach, and validate both the provided computational complexities of the individual algorithms, and the analysis of the approach as a whole.

The experiments were conducted with a variety of datasets of different types, sizes, and dimensionalities. The datasets used are also of different clusters distributions, and number of clusters. Each dataset comes with ground-truth labels such that each data instance is assigned a single label out of a set of predetermined labels. Our evaluation focuses on both the clustering accuracy and the running time of each algorithm. For all experiments, after the clustering was performed, the cluster labels were compared to the ground-truth labels, and the Normalized Mutual Information (NMI) [62] between clustering labels and the ground-truth labels was calculated. We also recorded the running time (in minutes) of the steps of each algorithm.

We conducted two sets of experiments. The first was conducted on a single node to assess the clustering accuracy of the proposed algorithms compared to both the previously proposed kernel k-means approximation and the exact kernel k-means algorithm. Further, we studied the effect of the tuning parameters on the clustering accuracy of each algorithm as a part of these one-node experiments. The second set of experiments was conducted using large-scale datasets in a distributed environment composed of 20 nodes. In these experiments, we have evaluated the scalability and performance of the proposed algorithms. Additionally, we have reported the clustering accuracy of each algorithm, to confirm the correctness and the effectiveness of the proposed approach when applied to very large datasets.

4.1 Single Node Experiments

4.1.1 Datasets

We evaluated the proposed algorithms by conducting experiments on four medium-scale datasets, called **USPS**, **PIE**, **ImageNet-50k**, and **MNIST**. The **PIE** dataset is a subset of 11,554 face images, in 68 classes, out of CMU PIE [63]. Both of the **USPS** and **MNIST** are handwritten digits in 10 classes and their sizes are 9,298 and 70,000 respectively [64]. The **ImageNet-50k** dataset is a subset of 50,000 images out of the 1,262,102 images of the Full-ImageNet dataset [65]. We used a preprocessed version of the original ImageNet dataset that was prepared by Chitta *et al.* [6] to evaluate their approximate kernel k-means approach. Each image was represented by a 900-dimensional vector based on SIFT descriptors computed using all the images of the full ImageNet dataset. Each instance of the **ImageNet-50k** is assigned to one of 164 classes. The properties of the datasets are summarized in Table 4.1. It should be noted that the four datasets are of different sizes, numbers of features, numbers of classes, and class distributions. Both **PIE** and **MNIST** are of balanced classes (i.e., classes are approximately of the same number of instances), while **USPS** and **ImageNet-50k** are of imbalanced classes, as shown in Figures 4.1 and 4.2.

Table 4.1: The properties of the datasets used in the single-node experiments.

Data set	Type	# Intstances	# Features	# Clusters
USPS	Digit Images	9,298	256	10
PIE	Face Images	11,554	4,096	68
ImageNet-50k	Images	50,000	900	164
MNIST	Digit Images	70,000	784	10

4.1.2 Setup

The single-node experiments focused on evaluating the effectiveness of the approximate kernel k-means algorithms proposed in this thesis, by comparing their clustering accuracy results to those of the exact kernel k-means and other recently proposed approximations. In addition, we extensively studied the effect of the proposed algorithms' tuning parameters on the clustering accuracy. We implemented cerntralized versions of the algorithms described in Chapter 3 -

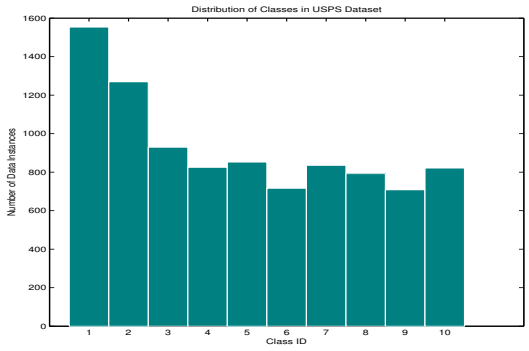


Figure 4.1: Number of data instances per class in the USPS dataset

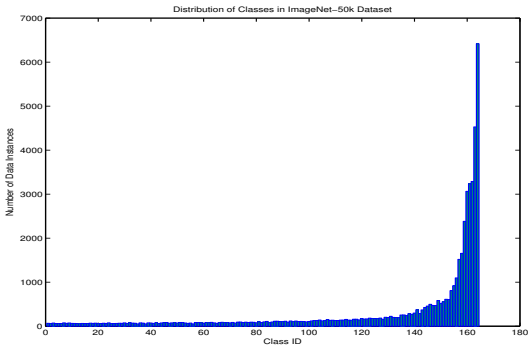


Figure 4.2: Number of data instances per class in the ImageNet-50k dataset

APNC via Nyström method (*APNC-Nys*) and APNC via Stable Distributions (*APNC-SD*) - using MATLAB on a single machine of 8GB RAM. We also compared them to the approximate Kernel k-means approach (*Approx KKM*) presented in [6], and the two Random Fourier Features-based algorithms (*RFF* and *SV-RFF*) presented in [7]. We used the MATLAB implementation provided by the authors of the *Approx KKM*, *RFF*, and *SV-RFF* approaches [1].

For *APNC-Nys*, *APNC-SD*, and *Approx KKM*, we used three different values for the number of samples l, while fixing the Gaussianity parameter t of *APNC-SD* to 100, and the embedding dimensionality m to 1000. For a fair comparison, we set the number of fourier features used in *RFF* and *SV-RFF* to 500, to obtain 1000-dimensional embeddings as in *APNC-SD*. An RBF kernel was used for both the **PIE** and **Imgnet-50k** datasets. The kernel width parameter σ was estimated for each dataset using the self-tuning method used in [46] and [6], where σ is computed as $\sigma = \rho\delta$ where ρ is set to the average euclidean distance between all pairs of data samples and δ was tuned to the value in $[0, 1]$ that achieved the best clustering accuracy. We used a neural kernel for the **USPS** data set and a polynomial kernel for the **MNIST** data set (the definition of each of the kernel functions used is given is Section 2.3). Following [6], the kernel parameters a, b, and d were set to $0.0045, 0.11$, and 5 respectively. Table 4.2 summarizes the average and standard deviation of the NMIs achieved in 20 different runs of each algorithm for each dataset, using different values for l. Being limited to only shift-invariant kernels, both *RFF* and *SV-RFF* were only used for the data sets **PIE** and **ImageNet-50k**. Since neither *RFF* nor *SV-RFF* uses a sample of data instance, their clustering accuracies are the same under different values for l. A t-test with 95% confidence level was used to determine the best performing algorithm for each dataset using each value for l. Further, Figure 4.3 shows the effect of the number of samples used

[1] Available at http://sites.google.com/site/radhacr/academics/projects/software

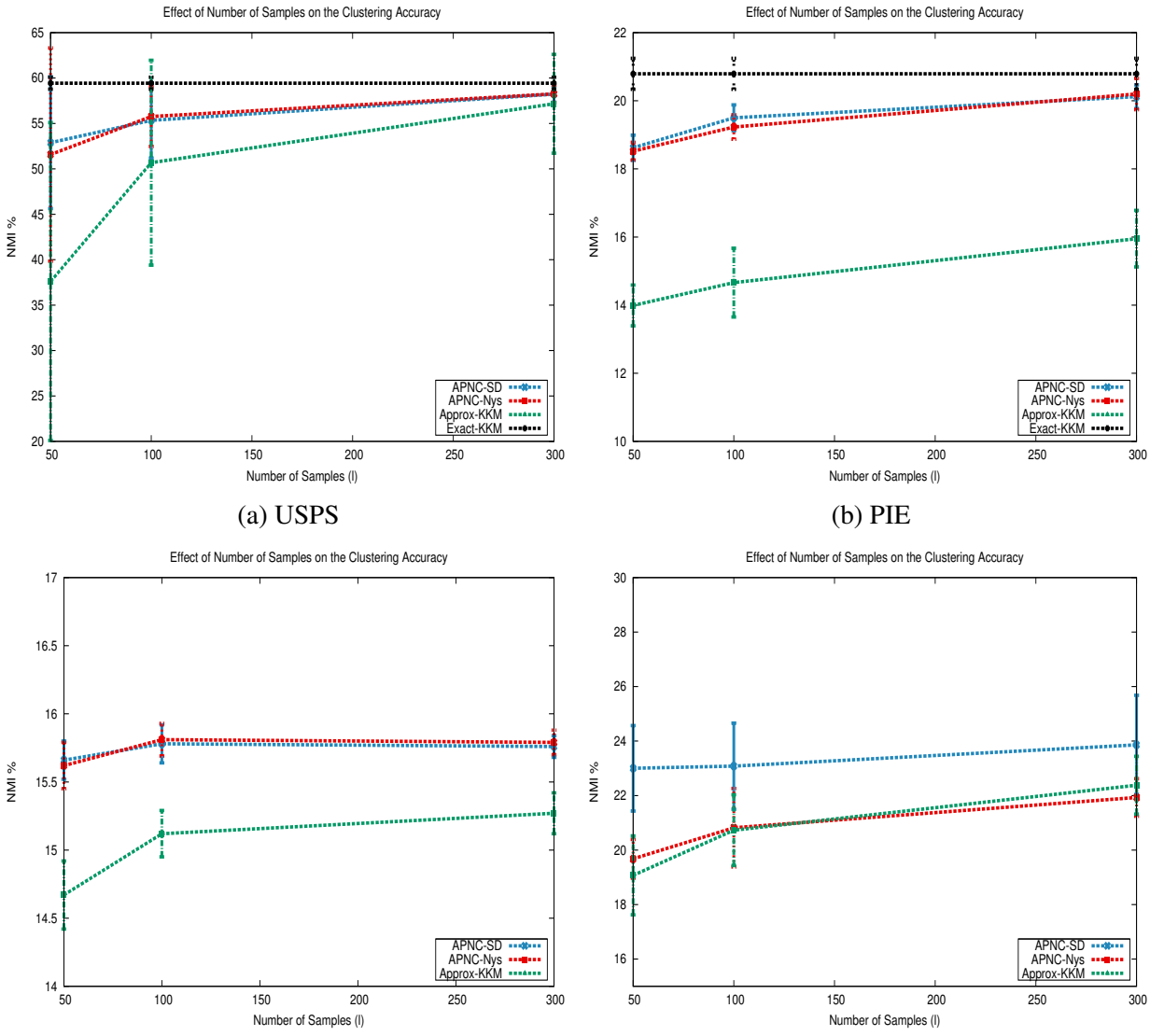
Table 4.2: The NMIs (%) of different kernel k-means approximations (single-node experiments). In each sub-table, the best performing approximation(s) for each l, (according to t-test with 95% confidence level) is highlighted in bold.

Methods	$l = 50$	$l = 100$	$l = 300$
	PIE - 11K, RBF		
RFF	5.2 ± 0.12	5.2 ± 0.12	5.2 ± 0.12
SV-RFF	5.15 ± 0.11	5.15 ± 0.11	5.15 ± 0.11
Approx KKM	13.99 ± 0.6	14.66 ± 1.01	15.95 ± 0.83
APNC-Nys	$\mathbf{18.52 \pm 00.26}$	$\mathbf{19.23 \pm 00.36}$	$\mathbf{20.20 \pm 00.46}$
APNC-SD	$\mathbf{18.62 \pm 0.37}$	$\mathbf{19.5 \pm 0.38}$	$\mathbf{20.12 \pm 0.35}$
Exact KKM	20.7915 ± 0.4542		
	ImageNet - 50K, RBF		
RFF	6.12 ± 0.04	6.12 ± 0.04	6.12 ± 0.04
SV-RFF	5.96 ± 0.06	5.96 ± 0.06	5.96 ± 0.06
Approx KKM	14.67 ± 0.25	15.12 ± 0.17	15.27 ± 0.15
APNC-Nys	$\mathbf{15.62 \pm 00.17}$	$\mathbf{15.81 \pm 00.12}$	$\mathbf{15.79 \pm 00.09}$
APNC-SD	$\mathbf{15.66 \pm 0.14}$	$\mathbf{15.78 \pm 0.14}$	$\mathbf{15.76 \pm 0.08}$
	USPS - 9K, Neural		
Approx KKM	37.60 ± 17.50	50.68 ± 11.28	$\mathbf{57.17 \pm 5.44}$
APNC-Nys	$\mathbf{51.58 \pm 11.74}$	$\mathbf{55.77 \pm 03.30}$	$\mathbf{58.26 \pm 00.95}$
APNC-SD	$\mathbf{52.88 \pm 7.25}$	$\mathbf{55.34 \pm 4.15}$	$\mathbf{58.22 \pm 0.87}$
Exact KKM	59.4367 ± 0.6591		
	MNIST - 70K, Polynomial		
Approx KKM	19.07 ± 1.45	20.73 ± 1.30	22.38 ± 1.06
APNC-Nys	19.68 ± 00.71	20.82 ± 01.44	21.93 ± 00.69
APNC-SD	$\mathbf{23.00 \pm 1.57}$	$\mathbf{23.08 \pm 1.58}$	$\mathbf{23.86 \pm 1.82}$

on the clustering accuracy achieved by *APNC-Nys*, *APNC-SD* and *Approx KKM* compared to the exact kernel k-means algorithm. Due to the large memory requirement of storing the kernel matrix, we only used the exact kernel k-means algorithm for the datasets **PIE** and **USPS**.

The second set of the single-node experiments was conducted to study the effect of the introduced tuning parameters on the clustering accuracy of the proposed algorithms. The first parameter we studied was the value of the embedding dimensionality m. In each of the two proposed algorithms (*APNC-Nys* and *APNC-SD*), we fixed the value of l to 300 and changed the value of m from 50 to 1000. The clustering accuracy at $m = 50$, $m = 250$, $m = 500$, and

$m = 1000$ were recorded in each of the four datasets. Figure 4.4 shows the average and standard deviation of the NMIs achieved in 20 different runs of *APNC-Nys* and *APNC-SD* in each dataset. It should be noted that in *APNC-Nys* the value of m cannot be greater than the value of l. So, at



(a) USPS

(b) PIE

(c) ImageNet-50k

(d) MNIST

Figure 4.3: Clustering accuracy of kernel k-means approximations using different numbers of samples l

$m = 500$ and $m = 1000$ we reported the best NMI achieved by *APNC-Nys* which was obtained at $m = l = 300$. Finally, the effect of the Gaussianity parameter t on the clustering accuracy of *APNC-SD* was studied by reporting the NMIs achieved at $t = 20$, $t = 50$, $t = 100$, and $t = 200$. The results obtained for each dataset are summarized in Figure 4.5.

4.1.3 Results

It can be observed from Table 4.2 that the centralized versions of the proposed algorithms (*APNC-Nys* and *APNC-SD*) were significantly superior to all the other kernel k-means approximations in terms of the clustering accuracy. Both methods performed similarly in all datasets except for the MNIST, in which *APNC-SD* outperformed the *APNC-Nys*. The poor performance of *RFF* and *SV-RFF* is consistent with the results of [31] that showed that for a fixed number of Fourier features, the approximation accuracy of RFF-based methods are determined by the properties of the eigenspectrum of the kernel matrix being approximated, which vary among different datasets. The table also shows that when using only 300 samples (i.e. $l = 300$), *APNC-Nys* and *APNC-SD* achieve very close clustering accuracy to that of the exact kernel k-means, which confirms the correctness and the reliability of the proposed approximations.

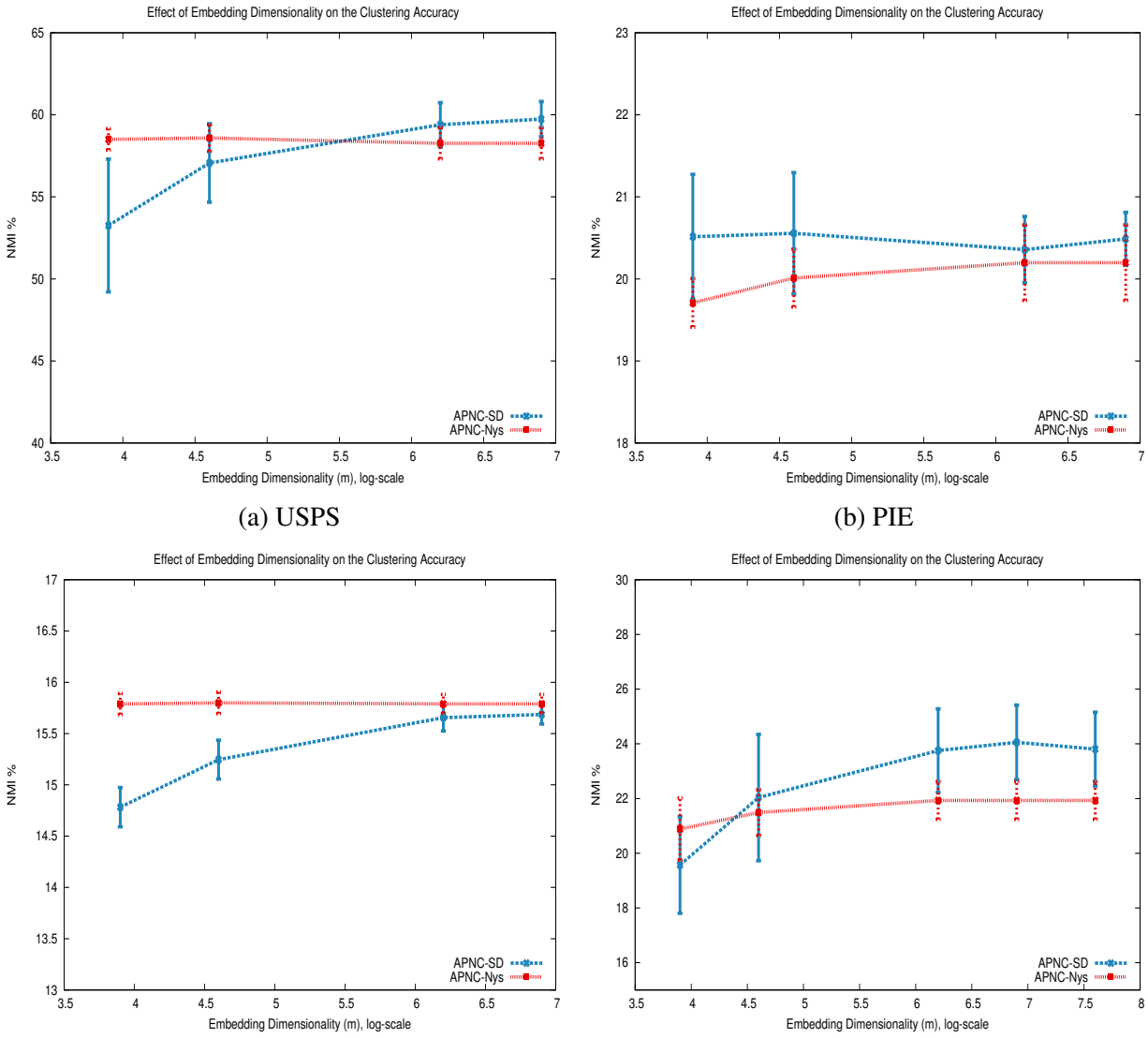
Figure 4.3 shows that increasing the number of samples used can improve the clustering accuracy of the proposed algorithms. The improvement rate decreases as l increases. Figure 4.4 compares the effect of increasing the embedding dimensionality on the clustering accuracy of *APNC-Nys* and *APNC-SD*. The figure shows that *APNC-Nys* is less sensitive to the value of m, which means that in *APNC-Nys* one can set m to very small values (e.g. $m = 50$) without worrying about the clustering accuracy. In addition, Figure 4.4 shows that setting m to a value greater than 500 does not contribute significantly to improving the clustering accuracy of *APNC-SD*. The effect of the Gaussianity parameter t on the clustering accuracy of *APNC-SD* seems to be quite minor, as can be noticed from Figure 4.5. Setting the value of t to 50 was sufficient to ensure the best clustering accuracy in most of the datasets.

4.2 Distributed Large-Scale Experiments

4.2.1 Datasets

The second part of the experiments was conducted using three real-world big datasets in a distributed environment of 20 nodes. The three datasets are called **RCV1**, **CovType**, and **ImageNet**. The **RCV1** dataset is a subset of 193,844 news documents in 103 categories prepared by Chen

et al. [53] to evaluate their distributed spectral clustering algorithms. Each document is represented by a very sparse vector of 47,236 tokens. The **CovType** dataset [66] is a set of 581,012 observations of cartographic variables in 7 classes. Each observation is associated with one of



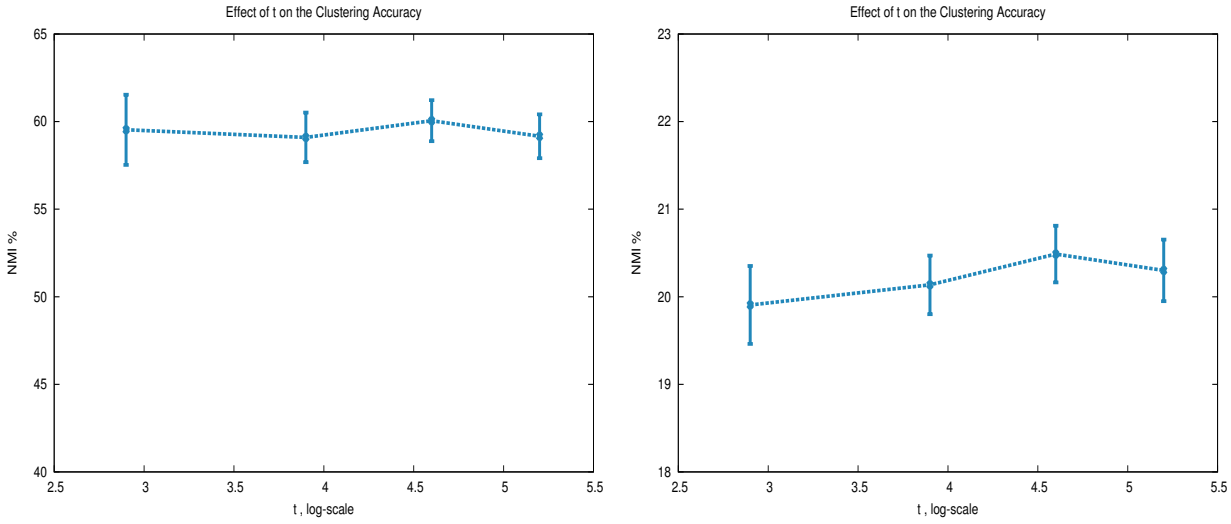
(a) USPS

(b) PIE

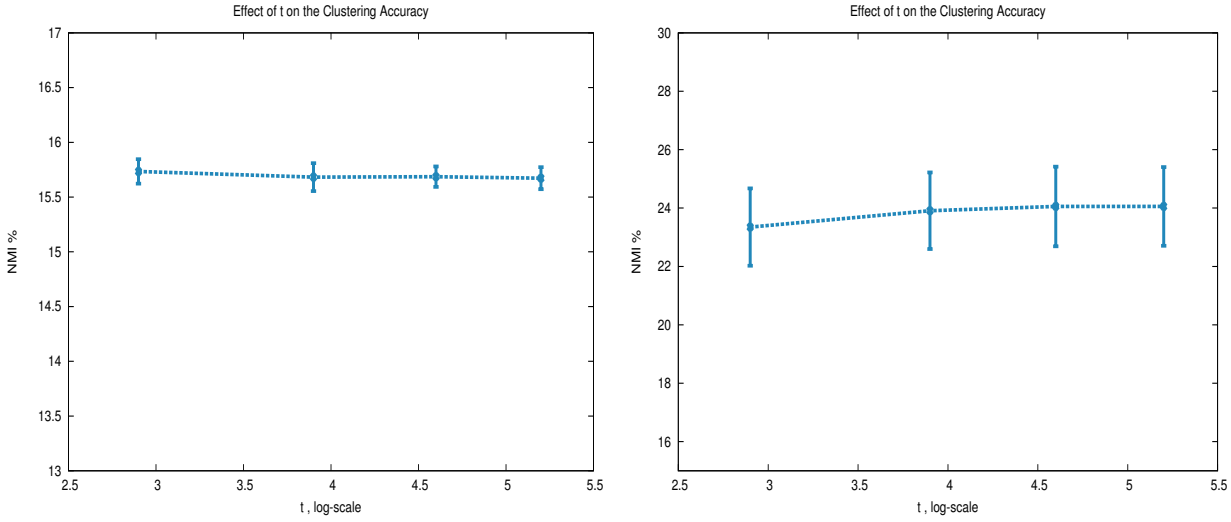
(c) ImageNet-50k

(d) MNIST

Figure 4.4: Clustering accuracy of APNC embeddings (APNC-SD and APNC-Nys) using different values for the target dimensionality m



(a) USPS (b) PIE



(c) ImageNet-50k (d) MNIST

Figure 4.5: Clustering accuracy of APNC embeddings via Stable Distributions (APNC-SD) using different values for the Gaussianity parameter t

seven possible forest cover types. The **CovType** dataset was used by Chitta *et al.* [7] and Chen *et al.* [46] to evaluate their large-scale kernel clustering approaches. We also used the full ImageNet dataset described in Section 4.1.1 above. The full ImageNet dataset (called **ImageNet**)

was also used by Chitta *et al.* [6] to evaluate their approximate kernel k-means approach. The properties of the datasets are summarized in Table 4.3. The three datasets are of three different types and characteristics. They are also of different numbers of classes and numbers of features. Their class distributions are plotted in Figures 4.6, 4.7, and 4.8.

Table 4.3: The properties of the data sets used in the large-scale experiments.

Data set	Type	# Intstances	# Features	# Clusters
RCV1	Documents	193,844	47,236	103
CovType	Multivariate	581,012	54	7
ImageNet	Images	1,262,102	900	164

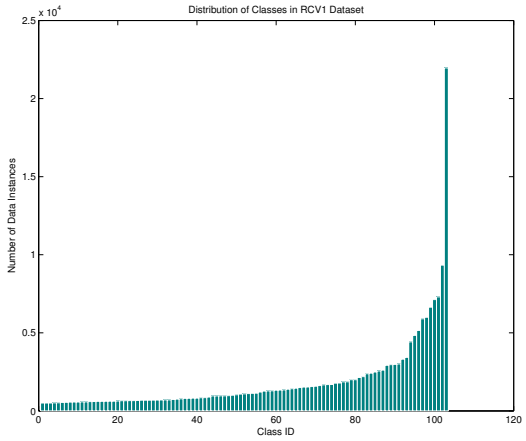


Figure 4.6: Number of data instances per class in the RCV1 dataset

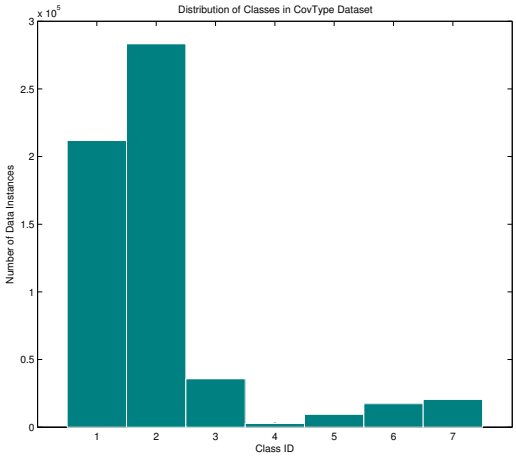


Figure 4.7: Number of data instances per class in the CovType dataset

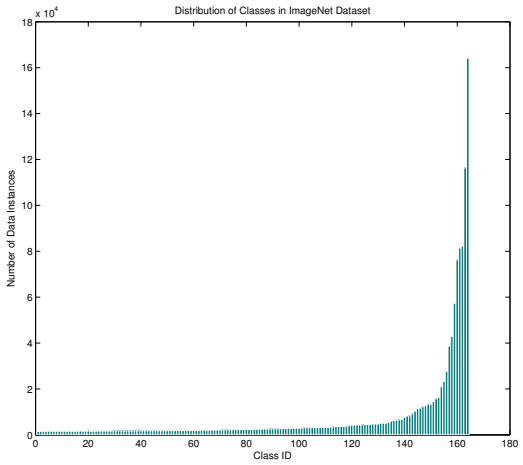


Figure 4.8: Number of data instances per class in the ImageNet dataset

4.2.2 Setup

The large-scale experiments were conducted on an Amazon EC2 [67] cluster which consists of 20 nodes (one master node and 19 workers). Each node had a 7.5GB RAM and a two-core processor. All instances were running Debian 6.0.5, Hadoop version 1.0.3, and Java 1.7.0. Each dataset was converted into a binary format in the form of a sequence of key-value pairs. Each pair represented a data instance where the key was set to the data instance ID and the value was the features vector corresponding to the data instance. A sparse representation for the features vectors of the **RCV1** dataset was used. That binary sequence of key-value pairs is the standard format used in Mahout [68] for storing distributed matrices on Hadoop File System (HDFS). Each dataset file was partitioned into 19 physical blocks of equal size, so that all blocks are processed in parallel after being written to HDFS.

We combined the proposed embedding methods *APNC-Nys* and *APNC-SD* with our parallelization strategy (the APNC embedding and the APNC clustering algorithms) and compared them to a baseline two-stage method (called *Two-Stage*) that uses the exact kernel k-means clustering results of a sample of l data instances to propagate the labels to all the other data instances [6]. The *Two-Stag* method is used as a sanity check to evaluate the relative improvement in clustering accuracy of the *APNC-Nys* and *APNC-SD*. *APNC-Nys* and *APNC-SD* were implemented using Hadoop, while the *Two-Stag* method was implemented using MATLAB on a single node, since we were only interested in its clustering accuracy. We evaluated the three algorithms using three different values for l, while fixing m in *APNC-Nys* and *APNC-SD* to $m = 500$, and t

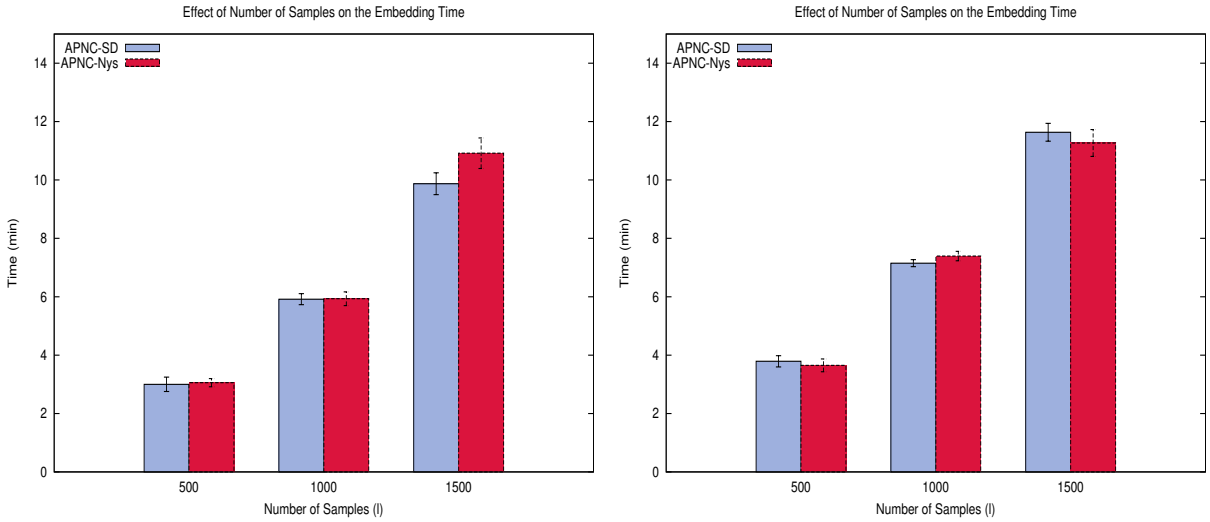
Table 4.4: The NMIs (%) of different kernel k-means approximations (large-scale experiments). In each sub-table, the best performing approximation(s) for each l, (according to t-test with 95% confidence level) is highlighted in bold.

Methods	$l = 500$	$l = 1000$	$l = 1500$
	RCV1 - 200K, RBF		
Two-Stag	13.33±00.53	13.56±00.53	13.56±00.06
APNC-Nys	**22.15±00.09**	**23.77±0.60**	**23.84±00.80**
APNC-SD	**22.21±00.39**	**24.34±00.26**	**23.55±00.17**
	CovType - 580K, RBF		
Two-Stag	08.95±02.98	10.23±01.07	09.85±01.88
APNC-Nys	09.53±02.55	12.31±00.74	12.51±01.08
APNC-SD	**15.96±01.03**	**15.08±01.40**	**15.56±00.18**
	ImageNet - 1.2M, RBF		
Two-Stag	07.51±00.42	07.58±00.21	07.71±00.20
APNC-Nys	**11.33±00.05**	**11.26±00.11**	**11.19±00.03**
APNC-SD	**11.27±00.06**	**11.26±00.04**	**11.10±00.05**

in *APNC-SD* to $t = 100$. We used a self-tuned RBF kernel [53] for all datasets. We used a fixed number of 20 iterations in the clustering step as a convergence criteria in both *APNC-Nys* and *APNC-SD*.

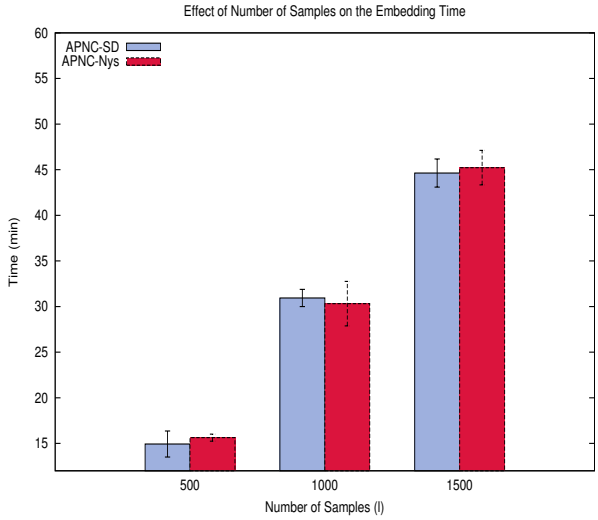
Table 4.4 summarizes the average and standard deviation of the NMIs achieved in three different runs of each algorithm. In addition, the time of the embedding step and the clustering step in each algorithm was recorded. Figure 4.9 shows the embedding time at each value of l in each algorithm when applied to the datasets **RCV1**, **CovType**, and **ImageNet**. The clustering time was not affected by the value of l. So, we only report the clustering time of each algorithm at $l = 500$. Table 4.5 summarizes the average and standard deviation of the clustering time of each dataset, using *APNC-Nys* and *APNC-SD*. The faster method according to a t-test with a 95% confidence level is highlighted in bold.

Another set of large-scale experiments was conducted to study the scalability of the proposed algorithms. We studied the scalability of *APNC-Nys* and *APNC-SD* as a function of the dataset size n, and of the embedding dimensionality m. We recorded the running time of the embedding and the clustering step of *APNC-Nys* and *APNC-SD* when applied to subsets of each of the datasets **RCV1** and **ImageNet**. We used subsets of four different sizes in each dataset. Each subset was parittioned into 19 blocks of equal size and distributed over the 20-node cluster. Figures 4.10 and 4.11 show the running time of the embedding step of *APNC-Nys* and *APNC-SD*



(a) RCV1

(b) CovType



(c) ImageNet

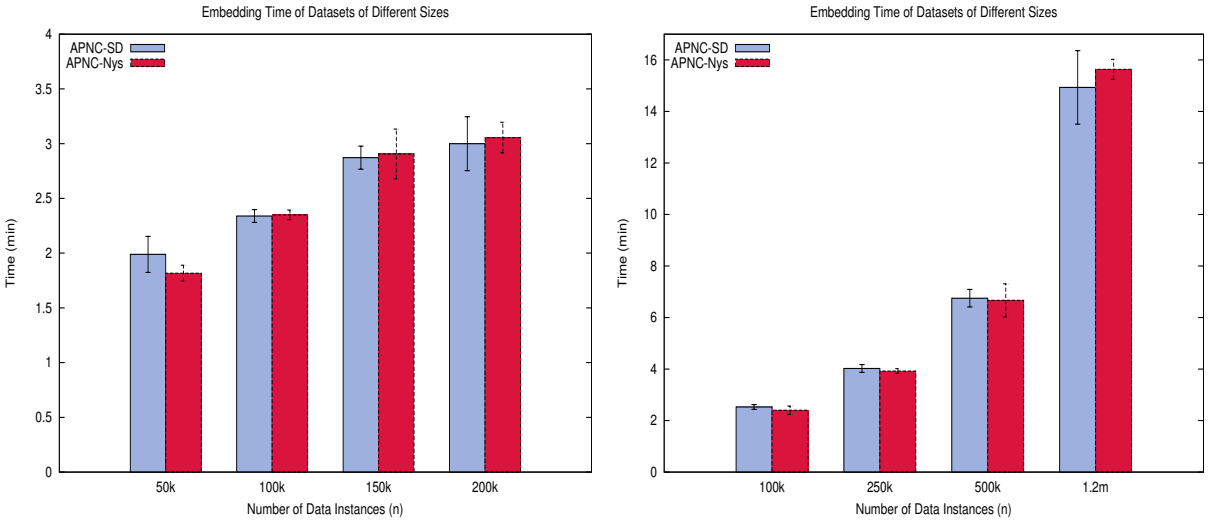
Figure 4.9: Embedding time of APNC embeddings via Nyström method (APNC-Nys) and APNC embeddings via stable distributions (APNC-SD) using different sample sizes l in different datasets

at different subsets of each of **RCV1** and **ImageNet**. Figures 4.12 and 4.13 show the running time of the clustering step of *APNC-Nys* and *APNC-SD* at different subsets of each of **RCV1** and

Table 4.5: The clustering times in minutes of different *APNC-Nys* and *APNC-SD*. For each dataset, the faster method according to t-test (with 95% confidence level) is highlighted in bold.

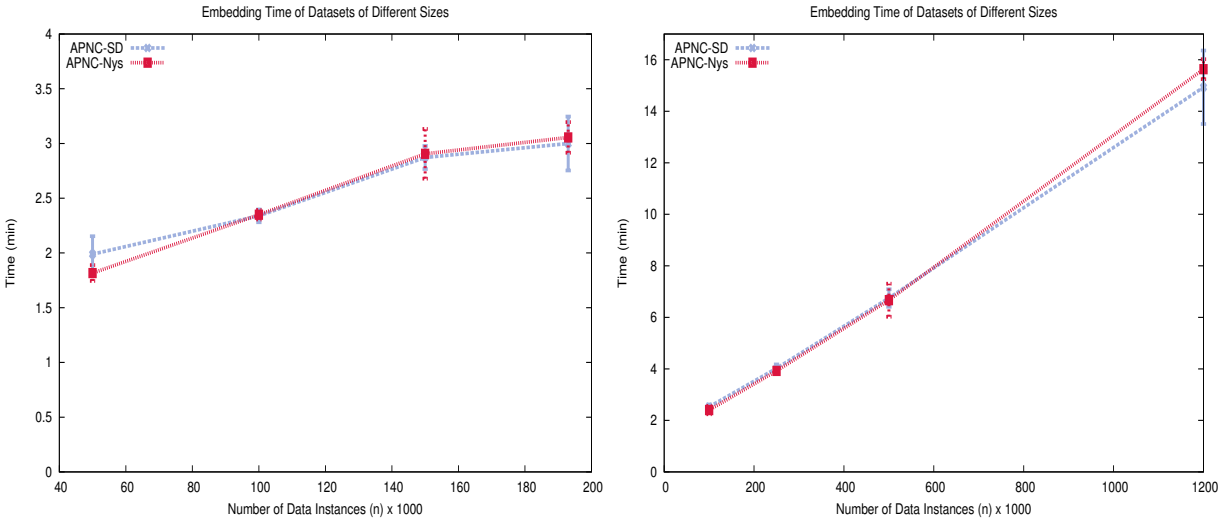
Datasets	**RCV1**	**CovType**	**ImageNet**
APNC-Nys	19.56 ± 0.07	16.44 ± 0.09	63.78 ± 2.75
APNC-SD	$\mathbf{15.59 \pm 0.44}$	$\mathbf{15.84 \pm 0.08}$	$\mathbf{23.73 \pm 0.31}$

ImageNet. Finally, for the full **RCV1** and **ImageNet** datasets, we recorded the embedding and the clustering time of *APNC-Nys* and *APNC-SD* using $m = 50$, $m = 250$, and $m = 500$, to study the scalability of the proposed algorithms when the embdding dimensionality is varied. Figures 4.14 and 4.15 show the embedding time and the clustering time of each dataset at different values of m. Also, we report the clustering accuracy of each dataset at the three values of m in Figure 4.16.



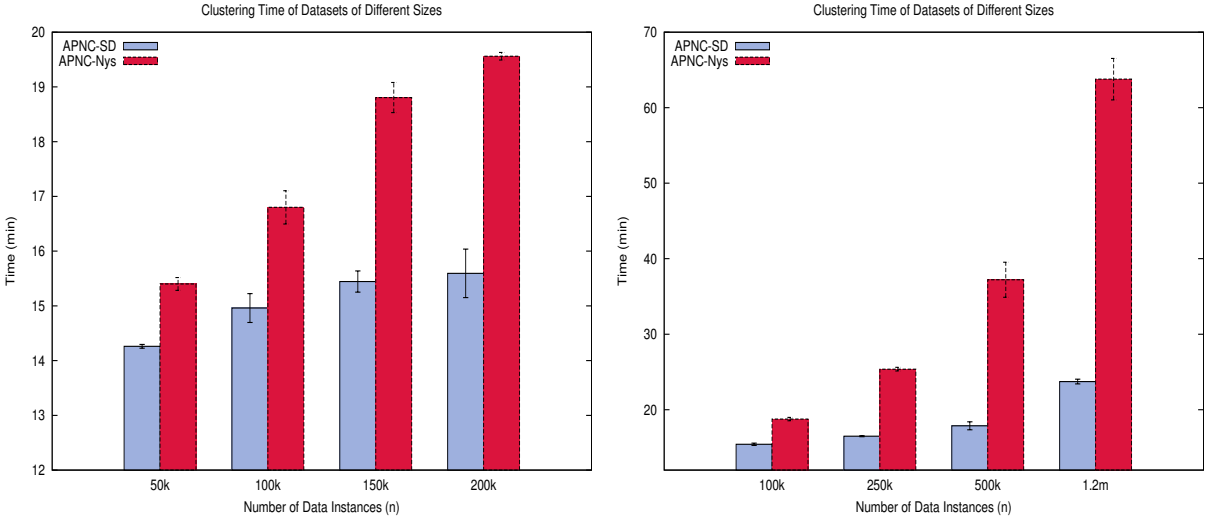
(a) RCV1 (b) ImageNet

Figure 4.10: Embedding time of APNC embeddings via Nyström method (APNC-Nys) and APNC embeddings via stable distributions (APNC-SD) using different dataset sizes



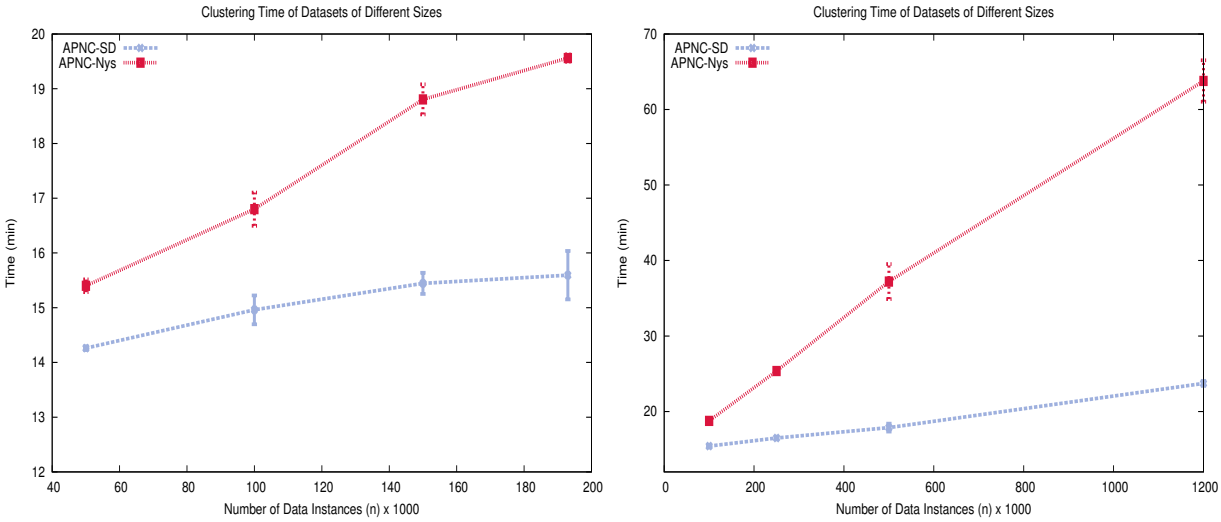
(a) RCV1 (b) ImageNet

Figure 4.11: The linear scalability of APNC embeddings



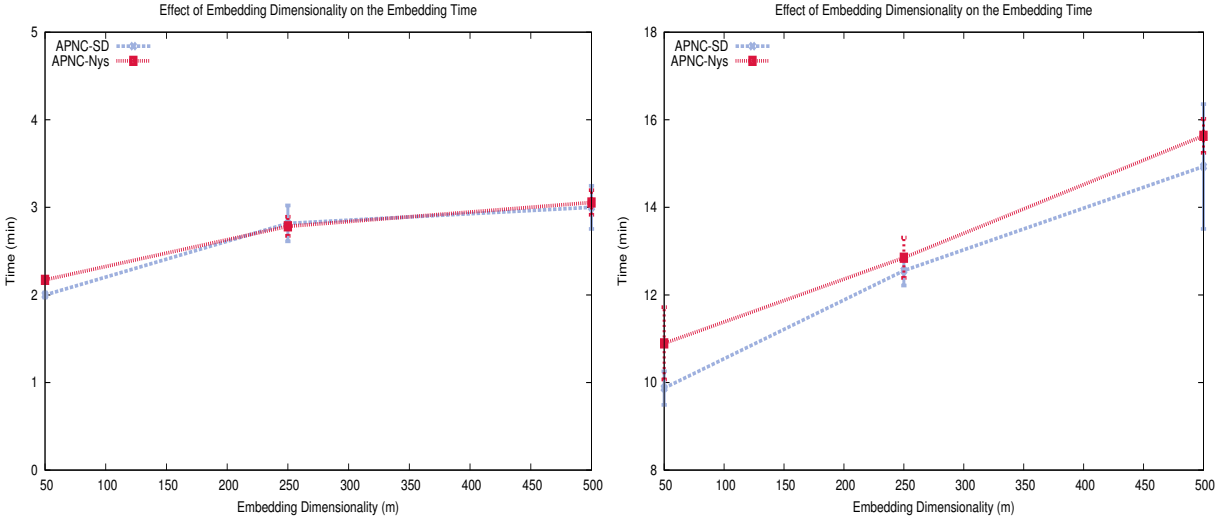
(a) RCV1 (b) ImageNet

Figure 4.12: Clustering time of APNC embeddings via Nyström method (APNC-Nys) and APNC embeddings via stable distributions (APNC-SD) using different dataset sizes



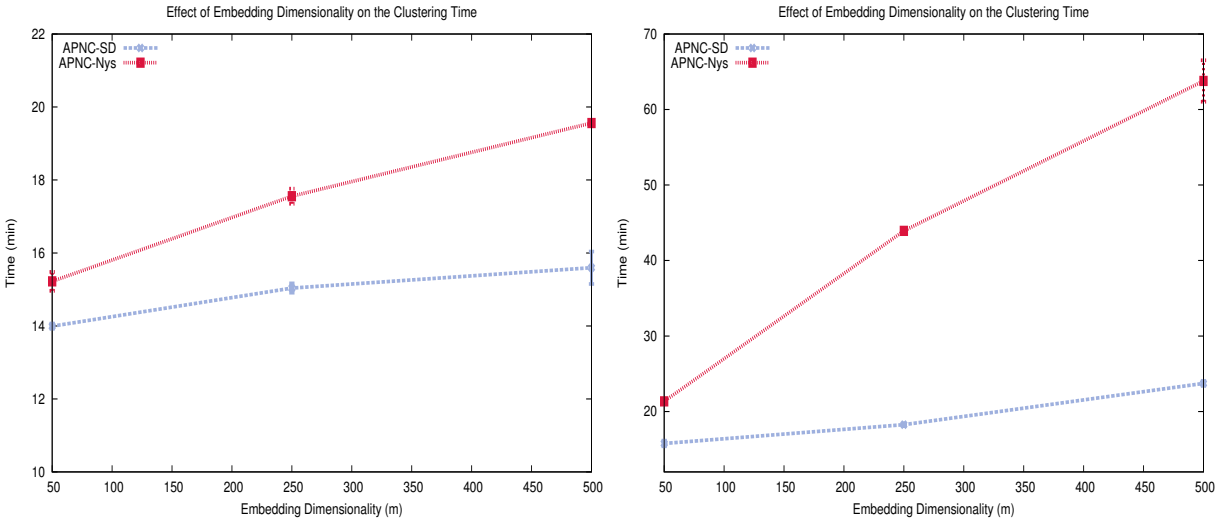
(a) RCV1 (b) ImageNet

Figure 4.13: The linear scalability of APNC clustering



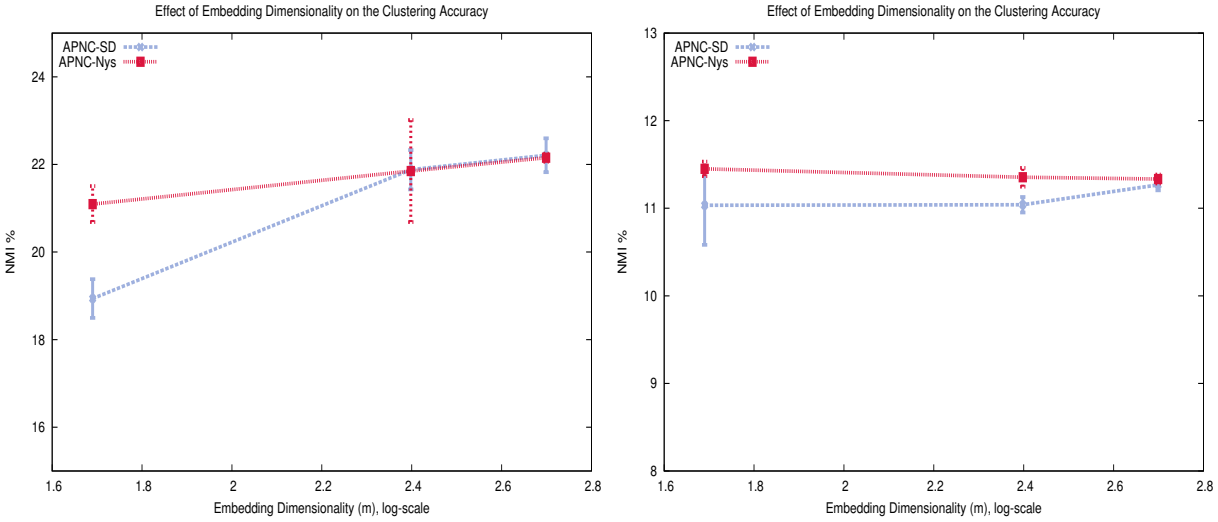
(a) RCV1 (b) ImageNet

Figure 4.14: The embedding time of APNC embeddings using different values for the embedding dimensionality (m)



(a) RCV1 (b) ImageNet

Figure 4.15: The clustering time of APNC embeddings using different values for the embedding dimensionality (m)



(a) RCV1 (b) ImageNet

Figure 4.16: The NMIs (%) of APNC embeddings using different values for the embedding dimensionality (m)

4.2.3 Results

The Clustering Accuracy

Table 4.4 demonstrates the effectiveness of the proposed algorithms in distributed settings compared to the baseline algorithm. Both *APNC-Nys* and *APNC-SD* achieved the same clustering accuracy in the **RCV1** and **ImageNet** datasets, while *APNC-SD* outperformed the *APNC-Nys* in the **CovType** dataset. It is also worth noting that, to the best of our knowledge, the best reported NMIs in the literature for the datasets **RCV1**, **CovType**, and **ImageNet** are 28.65% using the spectral clustering [53], 14% using *RFF* [7], and 10.4% using *Approx-KKM* of [6] respectively. Our algorithms managed to achieve better NMIs on both **CovType** and **ImageNet**, and a comparable clustering accuracy on **RCV1**. Table 4.4 also shows that the clustering accuracy can be improved by increasing the number of samples used for computing the embeddings in *APNC-Nys* and *APNC-SD*.

The Running Time

The running time of the proposed clustering approach consists of the time taken to compute the embedding, in addition to the time of running the APNC clustering algorithm on the computed embeddings.

Figure 4.9 shows that *APNC-SD* and *APNC-Nys* have comparable embedding times, which matches our analysis in Section 3.6. On the other hand, Table 4.5 shows that the clustering step of *APNC-SD* is faster than the clustering step of *APNC-Nys*, especially in the datasets with a large number of clusterings (**RCV1** and **ImageNet**). That advantage of the *APNC-SD* algorithm is from using the ℓ_1-distance as its discrepancy function while *APNC-Nys* uses the ℓ_2-distance as its discrepancy function.

To judge the overall efficiency of our algorithms, we compared the total clustering time on the **RCV1** dataset to the reported clustering running time of the same dataset on a 20-node cluster in [53]. We are unaware of any reported results for a distributed kernel k-means implementation. We are comparing our running times to the running times of the distributed spectral clustering of [53], to just get a sense of the efficiency of our algorithms. With $l = 1500$, the total clustering time using *APNC-SD* was on average 25 minutes, while the total clustering time of *APNC-Nys* was 29 minutes. The reported running time for the same data set on a 20-node cluster in [53] was 95 minutes. It is also worth noting that the reported clustering time for an images dataset of size $2,121,863$ on a 20-node cluster in [53] was 193 hours. Our approach was able to cluster $1,262,102$ images in only 38 minutes using the *APNC-SD* algorithm and 78 minutes using the

APNC-Nys. By the linear scalability of our algorithms (explained in the following section), the total clustering times of *APNC-SD* and *APNC-Nys* on that $2,121,863$ images dataset is expected to be 40 and 105 minutes respectively.

Scalability

The first part of our scalability study was concerned with the running time of the embedding and the clustering steps as the number of data instances increases. Figures 4.10 and 4.11 show the embedding time of four different subsets of the datasets **RCV1** and **ImageNet** with the number of samples l was fixed at 500, and the embedding dimensionality m set to 500. The figures show that *APNC-Nys* and *APNC-SD* have very close embedding times in all data subsets. Most importantly, the figures demonstrate the linear scalability of the embedding step of our approach (i.e., with a fixed number of nodes, the embedding time increases linearly with the dataset size). In terms of the clustering time, Figures 4.12 and 4.13 show that *APNC-SD* is always faster than *APNC-Nys*. The difference in the clustering times of *APNC-SD* and *APNC-Nys* becomes significant as the dataset size increases. The figures also confirms the linear scalability of the clustering step of both methods.

The second part of our scalability study focused on the effect of the embedding dimensionality m on the running time of the embedding and the clustering step. Figures 4.14 and 4.15 show that both of *APNC-Nys* and *APNC-Nys* scale linearly, in terms of both embedding and clustering time, as the value of m increases. Also, Figure 4.16 shows the effect of m on the clustering accuracy. The figure matches our results in the single-node experiments, which indicates that the clustering accuracy of *APNC-Nys* is less sensitive to the value of m than that of *APNC-SD*.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this thesis, we proposed distributed algorithms for scaling kernel k-means on MapReduce. We started by explaining the infeasibility of implementing the original kernel k-means algorithm or its previously proposed approximation on MapReduce in a scalable manner. Then, we defined a family of kernelized low-dimensional embeddings characterized by a set of computational and statistical properties. Based on those properties, we presented a unified parallelization strategy that first computes the corresponding embeddings of all data instances of the given dataset, then clusters them in a MapReduce-efficient manner. Next, we derived three embedding methods that were shown to adhere to the properties of the defined embedding family. The first embedding method is based on the Nström approximation; the second embedding method is an extension of the Nyström-based method, in which we exploited the recently proposed ensemble Nyström method to provide more accurate embeddings. The third embedding method was developed using the properties of the p-stable distributions. We analyzed all of the proposed algorithms in terms of their time, space, and network communication complexities. We used the analytical results obtained to prove the scalability and the efficiency of the proposed approach.

A comprehensive set of experiments was conducted to validate the correctness, efficiency, and scalability of the proposed approach and algorithms. We focused our experiments on the Nyström-based embedding and the p-stable distributions-based embedding. Combining each of the two embedding methods with the proposed parallelization strategy, we demonstrated the effectiveness of the presented algorithms by empirical evaluation on medium and large benchmark data sets. We studied the tuning parameters of each algorithm and the scalability of the approach

as a whole. The results obtained showed that our approach is able to provide an accurate kernel k-means approximation while being MapReduce-efficient and linearly scalable.

5.2 Future Work

Faster and More Memory-Efficient APNC Embeddings

In this thesis, we presented three instances of APNC embeddings with their corresponding MapReduce implementations, given in Algorithms 4, 5, and 7. We have shown that the key feature behind the scalability of the three instances is that their memory and time complexities are independent from the dataset size. However, there is still a room for developing faster and more memory-efficient APNC embeddings. As shown in Table 3.1, the three embeddings have quadratic memory requirements per node, and cubic time complexities either in the number of samples l, or in the target dimensionality m. By comparing the embedding times given in Figure 4.9 to the clustering times given in Table 4.5, it can be seen that 66% of the overall running time of the clustering algorithm was consumed only to computed the embeddings. One possible future direction of this work could be developing faster embeddings that adhere to the definition APNC embeddings given in Section 3.1. Reducing the memory requirements per node can also contribute to improving the scalability and the accuracy of the proposed approach. As shown in Figure 4.2, increasing the number of samples used can improve the clustering accuracy of APNC embeddings. However, the quadratic memory requirements per node of the three presented APNC instances will hinder increasing the value of l above a certain threshold. Developing APNC embeddings that are of much tighter memory requirements per node will facilitate using larger values for l when needed.

Spectral Clustering Extension

As pointed out by the authors in [53] and [55], the two main scalability bottlenecks of implementing the spectral clustering algorithm on MapReduce are (1) building the all-pairs similarity matrix, and (2) computing the eigenvectors of the corresponding graph Laplacian matrix. In Section 2.5.3, we briefly elaborated on the results presented by Dhillon *et al.* in [5] that showed the equivalence of the weighted kernel k-means and multiple variants of the spectral clustering. That equivalence was exploited by the authors in [5] to propose a multilevel approach to approximating the spectral clustering, using appropriate weighted kernel k-means variants. That multilevel approach is not suitable for MapReduce settings. One interesting extension of the approach introduced in this thesis might be to integrate it with the equivalence of the spectral clustering and

the kernel k-means, for developing a scalable and efficient spectral clustering algorithm that fits into the MapReduce model.

Theoretical Error Analysis

Besides our empirical comparison between the APNC via Nyström embeddings (*APNC-Nys*) and APNC via Stable Distributions embeddings (*APNC-SD*), one possible future direction is to theoretically compare both methods in terms of their respective approximation errors. That theoretical comparison might provide an explanation for the superiority of *APNC-SD* to *APNC-Nys* only in the **MNIST** (Table 4.2) and the **CovType** (Table 4.4) datasets, while being of similar clustering accuracies in all the other datasts. In addition, as explained in Section 2.4.1, the number of samples l used in the Nyström approximation corresponds to the cardinality of the subset of columns $\mathcal{L}$ sampled from the data matrix Φ (mapped to the kernel space), which provide a good approximation for Φ when projected to the span of $\mathcal{L}$. One the other hand, we have shown in Section 3.5 that the number of samples l used in *APNC-SD* is the sample size used to estimate the covariance matrix of the underlying distribution from which the columns of Φ are drawn. This suggests the following open questions. Which method requires less number of samples to achieve a certain clustering accuracy. Does the answer to that question depend on the dataset? Does it depend on the used kernel function? We believe that providing a theoretical error analysis for both methods might lead to a better understanding for how both methods compare to one another.

The kernel k-means algorithm is known to converge to a local optimal solution. But our approach uses different approximations for the kernel k-means algorithm depending on the embedding method used. It is also important to theoretically study the convergence of the kernel k-means when the approximations introduced in this thesis are employed.

MapReduce-Efficent Sampling

The accuracy of the Nyström approximation is determined by the selected samples of data instances that are used for computing the approximation. Several methods have been proposed for selecting a subset of instances that will improve the accuracy of the Nystrom approximation [34–36]. Studying the feasibility of implementing those methods on MapReduce in a scalable and efficient manner is an interesting future direction, as it might lead to improving the cluster accuracy of the Nyström-based embeddings presented in this thesis.

References

[1] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," *ACM Computing Surveys*, vol. 31, no. 3, pp. 264–323, 1999.

[2] I. Dhillon, Y. Guan, and B. Kulis, "Kernel k-means, spectral clustering and normalized cuts," in *ACM SIGKDD KDD*, 2004, pp. 551–556.

[3] U. von Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, 2007.

[4] B. Schölkopf and A. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.

[5] I. S. Dhillon, Y. Guan, and B. Kulis, "Weighted graph cuts without eigenvectors a multilevel approach," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 29, no. 11, pp. 1944–1957, 2007.

[6] R. Chitta, R. Jin, T. C. Havens, and A. K. Jain, "Approximate kernel k-means: Solution to large scale kernel clustering," in *ACM SIGKDD KDD*, 2011, pp. 895–903.

[7] R. Chitta, R. Jin, and A. K. Jain, "Efficient kernel clustering using random Fourier features," in *IEEE ICDM*, 2012, pp. 161–170.

[8] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, and A. Rabkin, "A view of cloud computing." *Communication of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[9] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[10] H. Karloff, S. Suri, and S. Vassilvitskii, "A model of computation for MapReduce," in *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '10. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2010, pp. 938–948.

[11] T. White, *Hadoop: The Definitive Guide*, 1st ed. O'Reilly Media, Inc., 2009.

[12] A. K. Farahat, A. Elgohary, A. Ghodsi, and M. S. Kamel, "Distributed Column Subset Selection on MapReduce," in *Proceedings of the Thirteenth IEEE International Conference on Data Mining*, 2013.

[13] A. Ene, S. Im, and B. Moseley, "Fast clustering using MapReduce," in *ACM SIGKDD KDD*, 2011, pp. 681–689.

[14] J. Lin and C. Dyer, *Data-Intensive Text Processing with MapReduce*, ser. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2010.

[15] U. Kang, C. Tsourakakis, A. Appel, C. Faloutsos, and J. Leskovec, "Hadi: Fast diameter estimation and mining in massive graphs with hadoop," *CMU-ML-08-117*, 2008.

[16] J. Xiang, C. Guo, and A. Aboulnaga, "Scalable maximum clique computation using mapreduce," in *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, 2013, pp. 74–85.

[17] C. Liu, H.-c. Yang, J. Fan, L.-W. He, and Y.-M. Wang, "Distributed nonnegative matrix factorization for web-scale dyadic data analysis on mapreduce," in *Proceedings of the 19th international conference on World wide web*, ser. WWW '10, 2010, pp. 681–690.

[18] S. Singh, J. Kubica, S. Larsen, and D. Sorokina, "Parallel large scale feature selection for logistic regression," *Proceedings of the SIAM International Conference on Data Mining*, pp. 1171–1182, 2009.

[19] X. Meng and M. Mahoney, "Robust regression on mapreduce," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 2013, pp. 888–896.

[20] B. Bahmani, K. Chakrabarti, and D. Xin, "Fast personalized pagerank on mapreduce," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, ser. SIGMOD '11, 2011, pp. 973–984.

[21] J.-G. Lee, J. Han, and K.-Y. Whang, "Trajectory clustering: A partition-and-group framework," in *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '07, 2007, pp. 593–604.

[22] A. Ben-Dor, R. Shamir, and Z. Yakhini, "Clustering gene expression patterns," *Journal of computational biology*, vol. 6, no. 3-4, pp. 281–297, 1999.

[23] B. Frey and D. Dueck, "Clustering by passing messages between data points," *Science*, vol. 315, no. 5814, p. 972, 2007.

[24] N. A. Yousri, M. S. Kamel, and M. A. Ismail, "A distance-relatedness dynamic model for clustering high dimensional data of arbitrary shapes and densities," *Pattern Recognition*, vol. 42, no. 7, pp. 1193–1209, 2009.

[25] H. Wang, S. Sen, A. Elgohary, M. Farid, M. Youssef, and R. R. Choudhury, "No need to war-drive: unsupervised indoor localization," in *Proceedings of the 10th international conference on Mobile systems, applications, and services*, ser. MobiSys '12, 2012, pp. 197–210.

[26] S. Lloyd, "Least squares quantization in PCM," *Information Theory, IEEE Transactions on*, vol. 28, no. 2, pp. 129–137, 1982.

[27] J. Mercer, "Functions of positive and negative type, and their connection with the theory of integral equations," *Philosophical Transactions of the Royal Society, London*, vol. 209, pp. 415–446, 1909.

[28] B. Kulis and K. Grauman, "Kernelized locality-sensitive hashing," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 34, no. 6, pp. 1092–1104, 2012.

[29] P. Drineas and M. W. Mahoney, "On the Nyström Method for approximating a Gram matrix for improved kernel-based learning," *Journal of Machine Learning Research*, vol. 6, pp. 2153–2175, 2005.

[30] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *NIPS*, 2007, pp. 1177–1184.

[31] T. Yang, Y.-F. Li, M. Mahdavi, R. Jin, and Z.-H. Zhou, "Nyström method vs random Fourier features: A theoretical and empirical comparison," in *NIPS*, 2012, pp. 485–493.

[32] C. Williams and M. Seeger, "Using the Nyström method to speed up kernel machines," in *NIPS*. MIT Press, 2000, pp. 682–688.

[33] A. K. Farahat, "Greedy representative selection for unsupervised data analysis," Ph.D. dissertation, University of Waterloo, 2012.

[34] P. Drineas, R. Kannan, and M. Mahoney, "Fast Monte Carlo algorithms for matrices II: Computing a low-rank approximation to a matrix," *SIAM Journal on Computing*, vol. 36, no. 1, pp. 158–183, 2007.

[35] S. Kumar, M. Mohri, and A. Talwalkar, "On sampling-based approximate spectral decomposition," in *Proceedings of the 26th International Conference on Machine Learning (ICML'09)*. New York, NY, USA: ACM, 2009, pp. 553–560.

[36] A. K. Farahat, A. Ghodsi, and M. S. Kamel, "A novel greedy algorithm for Nyström approximation." in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS'11)*, 2011, pp. 269–277.

[37] S. Kumar, M. Mohri, and A. Talwalkar, "Ensemble Nyström Method," in *NIPS*, 2009, pp. 1060–1068.

[38] M. Li, J. T. Kwok, and B.-L. Lu, "Making large-scale nyström approximation possible," in *ICML*, 2010, pp. 631–638.

[39] N. Halko, P.-G. Martinsson, Y. Shkolnisky, and M. Tygert, "An algorithm for the principal component analysis of large data sets," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2580–2594, 2011.

[40] A. Vedaldi and A. Zisserman, "Efficient additive kernels via explicit feature maps," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2010.

[41] P. Kar and H. Karnick, "Random feature maps for dot product kernels," in *AISTATS*, 2012, pp. 583–591.

[42] L. Hagen and A. Kahng, "New spectral methods for ratio cut partitioning and clustering," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 11, no. 9, pp. 1074–1085, 1992.

[43] J. Shi and J. Malik, "Normalized cuts and image segmentation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, no. 8, pp. 888–905, 2000.

[44] C. Ding, X. He, H. Zha, M. Gu, and H. Simon, "A min-max cut algorithm for graph partitioning and data clustering," in *Proceedings of the First IEEE International Conference on Data Mining (ICDM'01) 2001*, 2001, pp. 107–114.

[45] C. Fowlkes, S. Belongie, F. Chung, and J. Malik, "Spectral grouping using the nyström method," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 2, pp. 214–225, 2004.

[46] X. Chen and D. Cai, "Large scale spectral clustering with landmark-based representation," in *AAAI*, 2011, pp. 313–318.

[47] D. Yan, L. Huang, and M. Jordan, "Fast approximate spectral clustering," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'09)*. New York, NY, USA: ACM, 2009, pp. 907–916.

[48] A. Elgohary and M. Ismail, "Efficient data clustering over peer-to-peer networks," in *Intelligent Systems Design and Applications (ISDA), 2011 11th International Conference on*, nov. 2011, pp. 208 –212.

[49] S. Datta, C. Giannella, and H. Kargupta, "Approximate distributed k-means clustering over a peer-to-peer network," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 21, no. 10, pp. 1372–1388, 2009.

[50] E. Januzaj, H.-P. Kriegel, and M. Pfeifle, "Towards effective and efficient distributed clustering," in *In Workshop on Clustering Large Data Sets (ICDM*, 2003, pp. 49–58.

[51] G. Forman and B. Zhang, "Distributed data clustering can be efficient and exact," *SIGKDD Explor. Newsl.*, vol. 2, no. 2, pp. 34–38, Dec. 2000.

[52] K. M. Hammouda and M. S. Kamel, "Models of distributed data clustering in peer-to-peer environments," *Knowl. Inf. Syst.*, vol. 38, no. 2, pp. 303–329, 2014.

[53] W.-Y. Chen, Y. Song, H. Bai, C.-J. Lin, and E. Chang, "Parallel spectral clustering in distributed systems," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 33, no. 3, pp. 568 –586, 2011.

[54] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. Cambridge, MA: MIT Press, 1994.

[55] F. Gao, W. Abd-Almageed, and M. Hefeeda, "Distributed approximate spectral clustering for large-scale datasets," in *HPDC*. ACM, 2012, pp. 223–234.

[56] S. Papadimitriou and J. Sun, "Disco: Distributed co-clustering with map-reduce: A case study towards petabyte-scale end-to-end mining," in *Data Mining, 2008. ICDM '08. Eighth IEEE International Conference on*, 2008, pp. 512–521.

[57] R. L. Ferreira Cordeiro, C. Traina, Junior, A. J. Machado Traina, J. López, U. Kang, and C. Faloutsos, "Clustering very large multi-dimensional datasets with mapreduce," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '11, 2011, pp. 690–698.

[58] A. Elgohary, A. K. Farahat, M. S. Kamel, and F. Karray, "Approximate nearest centroid embedding for kernel k-means," big Learning Workshop, Advances in Neural Information Processing Systems (NIPS), 2013.

[59] ——, "Embed and conquer: Scalable embeddings for kernel k-means on mapreduce," in *The SIAM International Conference on Data Mining (SDM)*, 2014.

[60] ——, "Embed and conquer: Scalable embeddings for kernel k-means on mapreduce," *CoRR*, vol. abs/1311.2334, 2013.

[61] P. Indyk, "Stable distributions, pseudorandom generators, embeddings and data stream computation," in *Proceedings of the Symposium on Foundations of Computer Science*, 2000.

[62] A. Strehl and J. Ghosh, "Cluster ensembles—A knowledge reuse framework for combining multiple partitions," *Journal on Machine Learning Research*, vol. 3, pp. 583–617, 2003.

[63] T. Sim, S. Baker, and M. Bsat, "The cmu pose, illumination, and expression database," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, no. 12, pp. 1615–1618, Dec. 2003.

[64] C.-C. Chang and C.-J. Lin, "Libsvm: A library for support vector machines," *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, May 2011.

[65] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[66] K. Bache and M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[67] "Amazon elastic compute cloud (Amazon EC2)," http://aws.amazon.com/ec2/.

[68] "Apache mahout," http://mahout.apache.org.