Scalable Kernel Graph PCA for handling Big Data and its application to Clustering

Submitted by: Vishal Nemade

Supervisor: **Dr. Kapil Ahuja**

Department of Computer Science & Engineering
Indian Institute of Technology Indore

Submitted in partial fulfillment of requirements for the award of the degree
BACHELOR OF TECHNOLOGY

IIT Indore December 2017

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Submitted by: Vishal Nemade
December 2017

Certificate by BTP Guide

It is certified that the declaration made by the student is correct to the best of my knowledge and belief.

Dr. Kapil Ahuja
Associate Professor
Discipline of Computer Science & Engineering
Indian Institute of Technology Indore
(Project Guide)

Acknowledgements

I would like to thank my B.Tech Project supervisor **Dr.Kapil Ahuja** for his constant support in structuring the project and his valuable feedback throughout the course of this project. He gave me an opportunity to discover and work in such an interesting domain. His guidance proved really valuable in all the difficulties I faced in the course of this project.

I am really grateful to **Mr.Aditya Anand Shastri** who also provided valuable guidance and helped with the problems while working with Big Data and Apache Spark framework. He provided the initial pathway for stating the project in right manner and provided useful directions to proceed along whenever necessary.

I am also thankful to my family members,friends and colleagues who were a constant source of motivation. I am really grateful to Dept. of Computer Science & Engineering, IIT Indore for providing with the necessary hardware utilities to complete the project. I offer sincere thanks to everyone who else who knowingly or unknowingly helped me complete this project.

Vishal Nemade
140001039
Discipline of Computer Science & Engineering
Indian Institute of Technology Indore

Abstract

Kernel Principal Component Analysis (KPCA) is a dimension reduction method that is closely related to Principal Component Analysis (PCA). This report gives an overview of kernel PCA and presents an implementation of the method in MATLAB as well as Apache Spark. Also I propose an extension of KPCA i.e Kernel Graph PCA . The proposed algorithm is tested exhaustively on various data sets of different domain.The proposed algorithm is compared with Spectral Clustering.

'Big Data' is gaining momentum in every field under the sun. Large volumes of data are created in short periods of time. Clustering Big Data possess a huge problem when complexity of algorithm is high. In our algorithm the complexity is high as cube of n.

This report presents how the original Kernel based algorithms have been modified to make it scalable for handling Big Data. Apache Spark, which performs several times faster than conventional Big Data frameworks, has been used to implement the Scalable Algorithm. Further included is the discussion on various Big Data sampling methods namely K-Means Cluster Sampling and Bisecting K-Means Cluster Sampling which have been used in order to remove redundant samples thereby reducing clustering time. In the end, the results obtained on testing the scalable algorithm on various Big Datasets and the computational speedup obtained are presented.

Preface

This report on **"Scalable Kernel Graph PCA Clustering Algorithm for handling Big Data using Apache Spark"** is prepared under the supervision of Dr.Kapil Ahuja,Associate Professor,Computer Science & Engineering, IIT Indore.

Throughout this report, detailed description of the theoretical concepts used to develop and implement the Scalable Clustering Algorithm is provided.The novel clustering algorithm is the modification of Kernel PCA.It works better than Kernel PCA and Spectral Clustering.The Clustering algorithm is also implemented in Matlab and tested on various datasets from UCI repository. The implemented scalable algorithm is tested against two Big Datasets of varying sizes and the results are presented in a clear and concise manner. Further, detailed description of the cluster setup used to implement the classifier and instructions for implementing the same on Apache Spark framework are also provided in Appendix.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

This chapter highlights the background and motivation for the project. The problem statement of the project has been described and the importance of the results is also clearly portrayed. Towards the end of the chapter the objectives and expectation from this project are also outlined.

1.1 Background

Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters). It is a main task of exploratory data mining, and a common technique for statistical data analysis, used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, bioinformatics, data compression, and computer graphics.

In this report we would like to discuss the family of Kernel based clustering algorithms. Compared to the "traditional algorithms" such as k-means or single linkage, Kernel based clustering algorithms has many fundamental advantages. Results obtained by Kernel based clustering algorithms very often outperform the traditional approaches, Kernel based clustering algorithms is very simple to implement and can be solved efficiently by standard linear algebra methods.

The motivation behind this project is to develop a novel scalable Kernel based clustering algorithms for Clustering in Big Data Environment.

In the past decades, a group of methods known as dimension reduction methods have gained wide-spread attention. As the name implies, dimension reduction methods seek to solve this problem by reducing the number of dimensions of the data, while retaining most of the information in the data set. By removing redundant dimensions, it becomes easier

to recognize patterns or tendencies in the remaining data. Some examples of dimension reduction methods are Support Vector Machines and Random Projections (e.g. [11] and [2, 3]). In this report, one particular dimension reduction method is considered, namely Kernel Principal Component Analysis (KPCA) and other Kernel Graph Principal Component Analysis (KGPCA). The method is implemented in MATLAB and applied to data sets. In addition, different ways of modifying kernel PCA in order to improve the results for these particular data sets will be considered.

As regards Big Data, one of its signature traits is that large volumes are created in short periods of time. This data often comes from connected devices, such as mobile phones, vehicle fleets or industrial machinery. The reasons for generating and observing this data are many, yet a common problem is the clustering of Big Data[1]. 'Big Data' is gaining momentum in every field under the sun. Hence, it is essential to introduce such algorithms that work aptly for big data. For processing such tremendous volume of data, there is a need of big data frameworks such as Hadoop Map-Reduce, Apache Spark etc. Among these, Apache Spark performs upto 100 times faster than conventional frameworks like Hadoop Map-Reduce. For the effective analysis and interpretation of this data, scalable machine learning methods are required to overcome the space and time bottlenecks. This forms the basis of our motivation for the project.

Earlier work done in Kernel Algorithm for Big Data include application of Support Vector Data Description(SVDD)[8], Kernel Principal Component Analysis(KPCA)[13] and One-Class SVM (OCSVM). Kernel based approaches have shown promising results when used for Clustering as well as Classification.

1.2 Objective

As per the above discussion, the objective thus is the Development of a Scalable Kernel based Algorithm& its implementation on Apache Spark. The above objective has been divided into following goals:

- To design/enhance Kernel based algorithm to make it scalable for handling Big Data.
- To propose and develop Scalable algorithm for computing Gaussian Kernel and eigen value decomposition of large kernel matrix in distributed manner.
- To propose a modified Kernel based algorithm based on KPCA.
- To test the developed algorithm on various Big Datasets and Small Datasets.

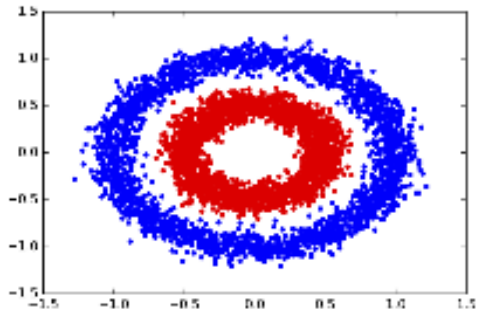
Chapter 2

Literature Review

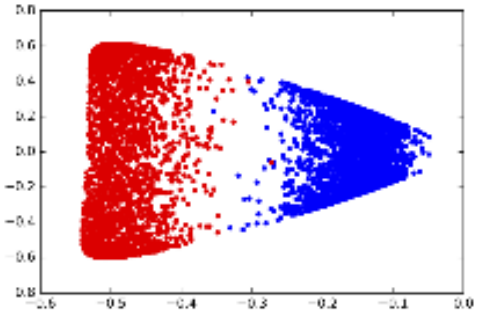
This chapter discusses the various concepts used during the course of this project. The chapter starts with discussion on Kernel based clustering algorithm and description of various methods of sampling used during this project. The end of the chapter comprises of discussion on Apache Spark, its features and advantages.

2.1 Kernel PCA

Kernel PCA uses the same basic idea as the statistical method Principal Com- ponent Analysis (PCA), namely that it seeks to project the set of data onto a low-dimensional subspace that captures the highest possible amount of variance in the data[3]. However, while PCA performs a linear separation of the data in the original space (referred to as R^d), kernel PCA embeds the data into a high dimensional space, called the feature space (referred to as F), by a mapping function Φ and performs a linear separation in that space instead. In this way also nonlinear separations of data are made possible. Thus, while kernel PCA looks for linear features in feature space, these features correspond to nonlinear features in the original lower dimensional space R^d , as shown in Figure 1. The data in the feature space is projected onto a low-dimensional subspace, spanned by the eigenvectors that capture most of the variance. One important fact is that it is not necessary to know the mapping Φ nor the feature space F explicitly in order to perform kernel PCA. Instead computations are performed on the inner product of pairs of points which are stored in a kernel matrix. The procedure of working with the data in feature space without knowing the mapping Φ nor F is known as ”the kernel trick” and is a central part of the kernel PCA method



(a) Kernel Kmeans



(b) Kernel Kmeans

2.1.1 A comparison of Kernel Kmeans and Kmeans

Despite its popularity, linear k-means clustering is not a universal solution to all clustering problems. In particular, linear k-means clustering strongly biases the recovered clusters towards isotropy and sphericity.

Kernel k-means clustering can correctly identify and extract a far more varied collection of cluster structures than the linear k-means clustering algorithm. However, kernel k- means clustering is computationally expensive when the non-linear feature map is high- dimensional and there are many input points.

2.1.2 Applications of Kernel Kmeans

Kernel Kmeans find wide use in the following areas:

- In dimensionality reduction of high dimension dataset.
- Datasets which are not linearly separable.
- For Protein , lung cancer and other biological datasets.
- Widely used in Image Processing Techniques

2.2 Spectral Clustering Algorithm

Spectral clustering[7] uses information obtained from the eig envalues and eigenvectors of their adjacency matrices for partitioning of graphs . It has many applications, such as in image segmentation (e.g. (Shi & Malik, 20 00)) and social network analysis (e.g. (Newman, Watts,& Strogatz, 2002)). The methods are called spectral, because they make use of the spectrum of the adjacency matrix of the data to cluster the points. Spectral clustering

algorithms have found an increasing following, especially after (Shi & Malik, 2000) and (Ng, Jordan, & Weiss, 2002). As opposed to k-means clustering, which results in convex sets, spectral clustering can solve problems, such as intertwined spirals, because it does not make assumptions on the form of the cluster. Given a sparse similarity graph, spectral clustering can be implemented efficiently even for large data sets (cf. (Verma & Meila, 2003)). Further advantages of learning about spectral clustering algorithms were noted as follows [12]:solution of clustering problems by standard linear algebra methods often more efficient than traditional algorithms (e.g. k-means, single linkage) A concise introduction into the field of spectral clustering can be found in (Luxburg, 2006), which this report relies on heavily. The comparison of algorithms herein after summarizes (Verma & Meila, 2003), a systematic comparison of spectral clustering algorithms, which demonstrate d that these complement and or compete with existing methods with convincing results. Most of the presented algorithms work in combination with clustering algorithms, such as k-means (the knowledge of which this report presupposes, refer to (MacKay, 2003) for an overview about clustering). Due to the subtle nature of the relationship between spectral parameters and the properties of datasets this report tries to outline the functioning of the algorithms in question rather than arguing for one of them as the best. Nevertheless comparison shows that some algorithms are more stable, have desirable properties, and have superior clustering results

2.2.1 Algorithm

2.3 Kernel Methods

In machine learning, kernel methods are a class of algorithms for pattern analysis, whose best known member is the support vector machine (SVM) [4]. The general task of pattern analysis is to find and study general types of relations (for example clusters, rankings, principal components, correlations, classifications) in datasets. For many algorithms that solve these tasks, the data in raw representation have to be explicitly transformed into feature vector representations via a user-specified feature map: in contrast, kernel methods require only a user-specified kernel, i.e., a similarity function over pairs of data points in raw representation.

Kernel methods[2] owe their name to the use of kernel functions, which enable them to operate in a high-dimensional, implicit feature space without ever computing the coordinates of the data in that space, but rather by simply computing the inner products between the images of all pairs of data in the feature space. This operation is often computationally cheaper than the explicit computation of the coordinates. This approach is called the "kernel

trick". Kernel functions have been introduced for sequence data, graphs, text, images, as well as vectors.

Reason for using Kernel

Kernels, are opposed to feature vectors. One big reason is that in many cases, computing the kernel is easy, but computing the feature vector corresponding to the kernel is really really hard. The feature vector for even simple kernels can blow up in size, and for kernels like the RBF kernel $k(x,y) = exp(-||x-y||^2)$ the corresponding feature vector is infinite dimensional. Yet, computing the kernel is almost trivial.

Many machine learning algorithms can be written to only use dot products, and then we can replace the dot products with kernels. By doing so, we don't have to use the feature vector at all. This means that we can work with highly complex, efficient-to-compute, and yet high performing kernels without ever having to write down the huge and potentially infinite dimensional feature vector. Thus if not for the ability to use kernel functions directly, we would be stuck with relatively low dimensional, low-performance feature vectors.

2.3.1 Gaussian Kernel

For the purpose of this thesis, Gaussian Kernel is being used. In machine learning, the (Gaussian) radial basis function kernel, or Gaussian kernel, is a popular kernel function used in various kernelized learning algorithms. In particular, it is commonly used in support vector machine classification.

The Gaussian kernel on two samples x and x', represented as feature vectors in some input space, is defined as [11]:

$$K(x,x') = exp(-\frac{||x-x'||^2}{2\sigma^2})$$

$||x-x'||^2$ may be recognized as the squared Euclidean distance between the two feature vectors. σ is a free parameter. An equivalent, but simpler, definition involves a parameter $\gamma = \frac{1}{2\sigma^2}$

$$K(x,x') = exp(-\gamma||x-x'||^2)$$

Since the value of the Gaussian kernel decreases with distance and ranges between zero (in the limit) and one (when x = x'), it has a ready interpretation as a similarity measure.

2.4 Sampling Big Data

Data sampling is a statistical analysis technique used to select, manipulate and analyze a representative subset of data points in order to identify patterns and trends in the larger data set being examined.

Sampling allows data scientists, predictive modelers and other data analysts to work with a small, manageable amount of data in order to build and run analytical models more quickly, while still producing accurate findings[5]. Sampling can be particularly useful with data sets that are too large to efficiently analyze in full – for example, in big data analytics applications. An important consideration, though, is the size of the required data sample. In some cases, a very small sample can tell all of the most important information about a data set. In others, using a larger sample can increase the likelihood of accurately representing the data as a whole, even though the increased size of the sample may impede ease of manipulation and interpretation. Either way, samples are best drawn from data sets that are as large and close to complete as possible.

Sampling Methods can be classified into one of two categories:

- **Probability Sampling**: Sample has a known probability of being selected
- **Non-probability Sampling**: Sample does not have known probability of being selected as in convenience or voluntary response surveys

Probability Sampling

In probability sampling it is possible to both determine which sampling[14] units belong to which sample and the probability that each sample will be selected. The following sampling methods are examples of probability sampling:

- Simple Random Sampling (SRS)
- Stratified Sampling
- Cluster Sampling
- Systematic Sampling
- Multistage Sampling (in which some of the methods above are combined in stages)

For the purpose of this thesis Cluster Sampling methods have been employed. Brief description of how both have been used is below.

Cluster Sampling

In this, the input samples are divided into groups called clusters and one representative from each group is taken as input sample.The clusters are mutually exclusive and collectively exhaustive.For the purpose of this thesis, K-Means & Bisecting K-Means clustering techniques are used to obtain cluster centers which are treated as input samples.

K Means Clustering

k-means clustering is a method of vector quantization [10], originally from signal processing, that is popular for cluster analysis in data mining. k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells.

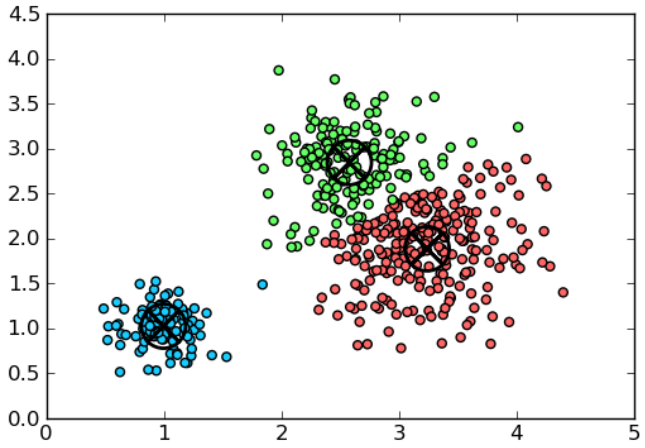


Fig. 2.2 K-Means Clustering

Bisecting K-Means Clustering

Bisecting K-means can often be much faster than regular K-means, but it will generally produce a different clustering[9].Bisecting k-means is a kind of hierarchical clustering. Hierarchical clustering is one of the most commonly used method of cluster analysis which seeks to build a hierarchy of clusters.

As Bisecting k-means is based on k-means, it keeps the merits of k-means and also has some advantages over k-means. First, bisecting k-means is more efficient when ‘k’ is large. For the k-means algorithm, the computation involves every data point of the data

set and k centroids. On the other hand, in each Bisecting step of Bisecting k-means, only the data points of one cluster and two centroids are involved in the computation. Thus, the computation time is reduced. Secondly, Bisecting k-means produce clusters of similar sizes, while k-means is known to produce clusters of widely different sizes

2.5 Big Data Handling Framework - Apache Spark

Apache Spark is an open-source cluster-computing framework. Originally developed at the University of California, Berkeley's AMPLab, the Spark codebase was later donated to the Apache Software Foundation, which has maintained it since. Spark provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. Apache Spark is a lightning-fast cluster computing technology, designed for fast computation. It is based on Hadoop MapReduce and it extends the MapReduce model to efficiently use it for more types of computations, which includes interactive queries and stream processing [6]. The main feature of Spark is its in-memory cluster computing that increases the processing speed of an application.

2.5.1 Features of Apache Spark

Apache Spark has following features.

- **Speed** - Spark helps to run an application in Hadoop cluster, up to 100 times faster in memory, and 10 times faster when running on disk. This is possible by reducing number of read/write operations to disk. It stores the intermediate processing data in memory.
- **Supports multiple languages** - Spark provides built-in APIs in Java, Scala, or Python. Therefore, you can write applications in different languages. Spark comes up with 80 high-level operators for interactive querying.
- **Advanced Analytics** - Spark not only supports 'Map' and 'reduce'.It also supports SQL queries, Streaming data, Machine learning (ML), and Graph algorithms.

2.5.2 Apache Spark Architecture

The Spark project stack currently is comprised of Spark Core and four libraries that are optimized to address the requirements of four different use cases. Individual applications will typically require Spark Core and at least one of these libraries. Spark's flexibility and

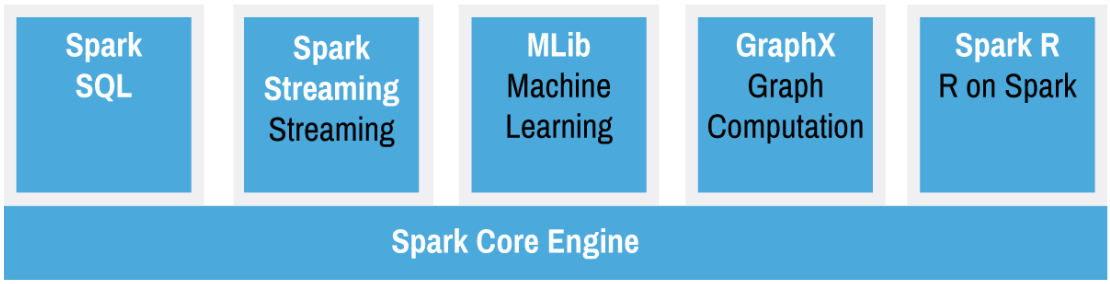


Fig. 2.3 Spark Architecture

power become most apparent in applications that require the combination of two or more of these libraries on top of Spark Core.These libraries are :

- Spark SQL
- Spark Streaming
- MLlib
- GraphX
- Spark R

Spark Built on Hadoop

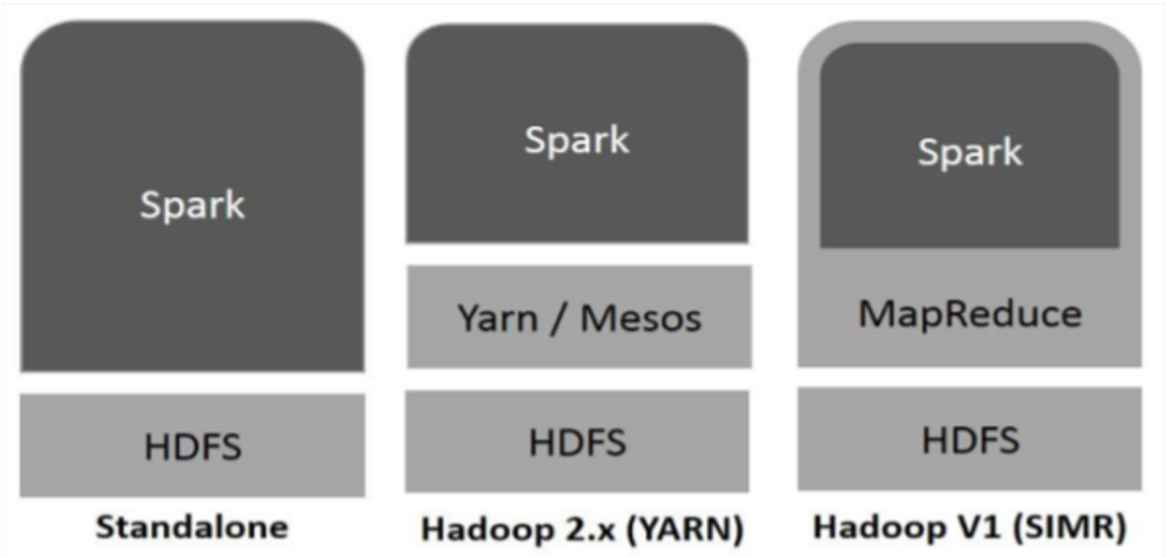


Fig. 2.4 Spark Deployment Methods

There are three ways of Spark deployment as explained below.

- **Standalone** - Spark Standalone deployment means Spark occupies the place on top of HDFS(Hadoop Distributed File System) and space is allocated for HDFS, explicitly. Here, Spark and MapReduce will run side by side to cover all spark jobs on cluster.
- **Hadoop Yarn** - Hadoop Yarn deployment means, simply, spark runs on Yarn without any pre-installation or root access required. It helps to integrate Spark into Hadoop ecosystem or Hadoop stack. It allows other components to run on top of stack.
- **Spark in MapReduce (SIMR)** - Spark in MapReduce is used to launch spark job in addition to standalone deployment. With SIMR, user can start Spark and uses its shell without any administrative access.

2.5.3 Resilient Distributed Dataset (RDD)

The Resilient Distributed Dataset is a concept at the heart of Spark. It is designed to support in-memory data storage, distributed across a cluster in a manner that is demonstrably both fault-tolerant and efficient. Fault-tolerance is achieved, in part, by tracking the lineage of transformations applied to coarse-grained sets of data. Efficiency is achieved through parallelization of processing across multiple nodes in the cluster, and minimization of data replication between those nodes. Once data is loaded into an RDD, two basic types of operation can be carried out:

- **Transformations** which create a new RDD by changing the original through processes such as mapping, filtering, and more;
- **Actions** such as counts, which measure but do not change the original data. The original RDD remains unchanged throughout. The chain of transformations from RDD1 to RDDs are logged, and can be repeated in the event of data loss or the failure of a cluster node.

Transformations are said to be lazily evaluated, meaning that they are not executed until a subsequent action has a need for the result. This will normally improve performance, as it can avoid the need to process data unnecessarily. It can also, in certain circumstances, introduce processing bottlenecks that cause applications to stall while waiting for a processing action to conclude.

Where possible, these RDDs remain in memory, greatly increasing the performance of the cluster, particularly in use cases with a requirement for iterative queries or processes.

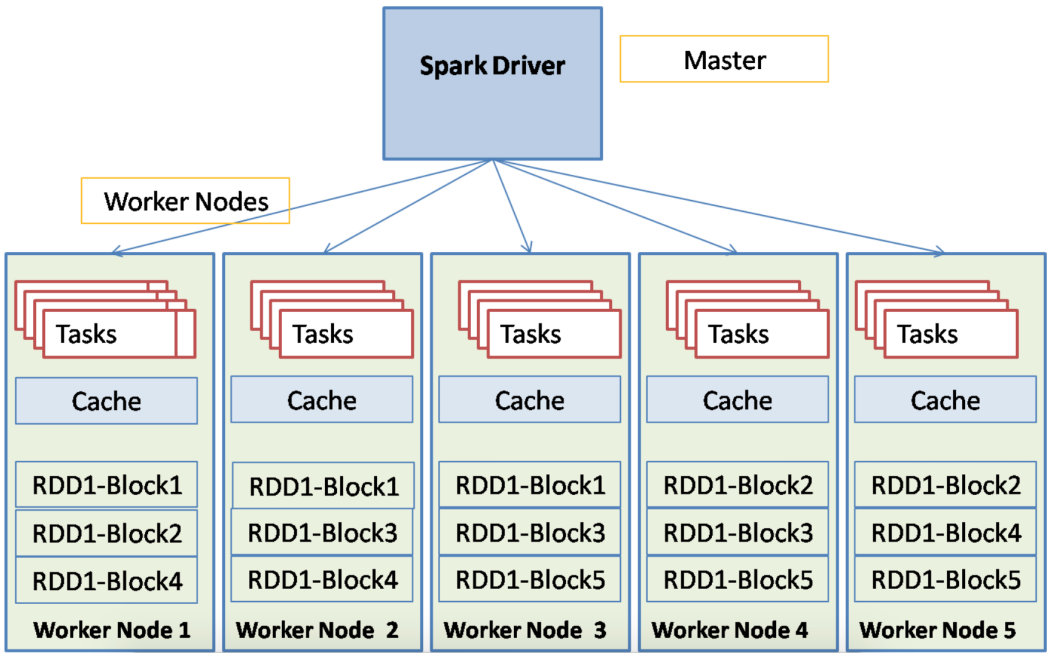


Fig. 2.5 Apache Spark RDDs

There are two ways to create RDDs - parallelizing an existing collection in your driver program, or referencing a dataset in an external storage system, such as a shared file system, HDFS, HBase, or any data source offering a Hadoop Input Format.

Spark makes use of the concept of RDD to achieve faster and efficient MapReduce operations

Data Sharing using Spark RDD

Data sharing is slow in MapReduce due to replication, serialization, and disk IO. Most of the Hadoop applications, they spend more than 90% of the time doing HDFS read-write operations.

Recognizing this problem, researchers developed a specialized framework called Apache Spark. The key idea of spark is Resilient Distributed Datasets (RDD); it supports in-memory processing computation. This means, it stores the state of memory as an object across the jobs and the object is sharable between those jobs. Data sharing in memory is 10 to 100 times faster than network and Disk.

The section below discusses how the iterative and interactive operations take place in Spark RDD.

Iterative Operations on Spark RDD

The illustration given below shows the iterative operations on Spark RDD. It will store intermediate results in a distributed memory instead of Stable storage (Disk) and make the system faster.

If the Distributed memory (RAM) is not sufficient to store intermediate results (State of the JOB), then it will store those results on the disk.

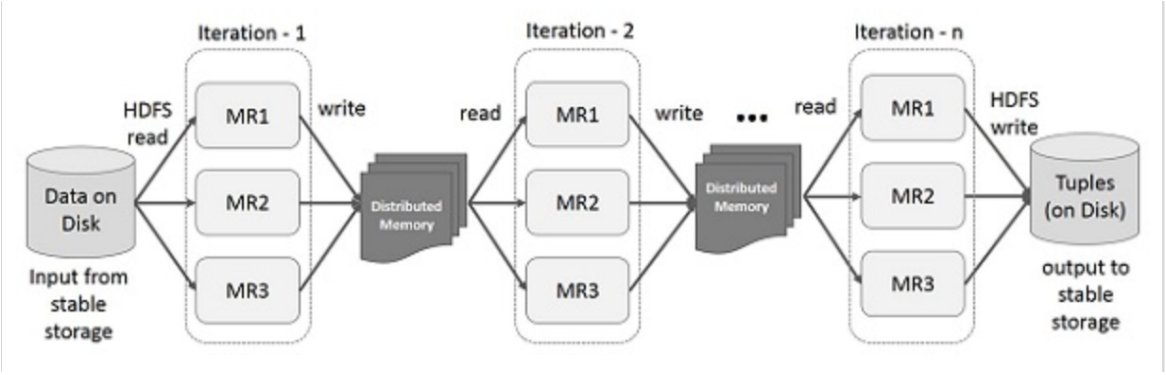


Fig. 2.6 Iterative Operations on Spark RDD

Interactive Operations on Spark RDD

This illustration shows interactive operations on Spark RDD. If different queries are run on the same set of data repeatedly, this particular data can be kept in memory for better execution times.

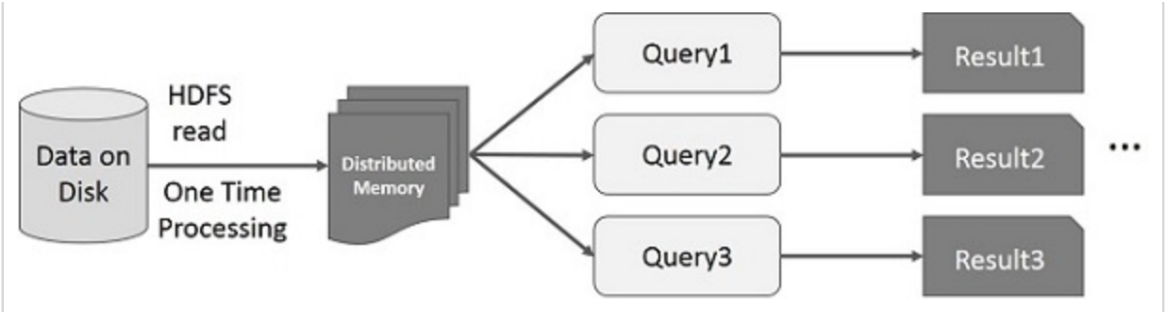


Fig. 2.7 Interactive Operations on Spark RDD

By default, each transformed RDD may be recomputed each time you run an action on it. However, you may also persist an RDD in memory, in which case Spark will keep the elements around on the cluster for much faster access, the next time you query it. There is also support for persisting RDDs on disk, or replicated across multiple nodes.

Chapter 3

Design of Scalable Kernel Graph PCA for Clustering

In this chapter, we will see the design and implementation of the Scalable Kernel Graph PCA which is based on Kernel PCA[14]. This includes discussion on modifications to kernel & computation algorithms and hence the overall modified algorithm. This chapter also discusses the implementation of the sampling methods on Apache Spark

3.1 Design of Scalable Gaussian Kernel Algorithm

The first step in Kernel based Clustering algorithm is the computation of Kernel Matrix.The RBF Kernel Matrix is similar to the Gaussian Kernel Matrix . The only difference is inclusion of sigma in Gaussian Kernel Matrix. The RBF Kernel matrix K for a set of N samples is a $N \times N$ symmetric matrix where the matrix entry K_{ij} represents the similarity measure between the i^{th} and j^{th} sample. The algorithm to compute the kernel matrix is discussed below:-

Input :

Samples[] $\rightarrow$ List of Samples

RDD[Samples[]] $\rightarrow$ RDD of Samples

Output :

KernelMatrix $\rightarrow$ Distributed Kernel Matrix

function : ComputeKernelMatrix(Samples[],RDD[Samples])

rowsRDD = emptyRDD()

foreach X in Samples[] do:

row = ComputeKernelMatrixRow(X,RDD[Samples[]])

```
            rowsRDD.union(row)
        KernelMatrix = RddToDistributedMatrix(rowsRDD)
        return KernelMatrix
end function
```

The function *ComputeKernelMatrixRow()* is as follows:

```
function : ComputeKernelMatrixRow(Sample X,RDD[Samples] rdd)
        rowRDD = rdd.map(Y => exp(-sqdist(X,Y))
        return rowRDD
end function
```

3.1.1 Eigen Value Decomposition using BLAS & LAPACK libraries

The BLAS (Basic Linear Algebra Subprograms) are routines that provide standard building blocks for performing basic vector and matrix operations. Because the BLAS are efficient, portable, and widely available, they are commonly used in the development of high quality linear algebra software, LAPACK for example, which provide significant performance improvements.

LAPACK is written in Fortran 90 and provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems.

Pre-built version of Apache Spark doesn't include BLAS and LAPACK. Hence, Spark was recompiled on all system of the cluster to include these libraries. The SVD along with LAPACK provides significant performance improvements.

3.2 Implementation of Sampling Methods

As per the discussion in the previous chapter, to reduce the size of Big Dataset, two sampling techniques, namely K-Means clustering and Bisecting K-Means clustering have been applied. The implementation procedures of each are discussed below.

3.2.1 K Means Clustering

K-means is one of the most commonly used clustering algorithms that clusters the data points into a predefined number of clusters. The *spark.mllib* implementation includes a parallelized

variant of the k-means++ method called kmeans||. The implementation in spark.mllib has the following parameters:

- k is the number of desired clusters. Note that it is possible for fewer than k clusters to be returned, for example, if there are fewer than k distinct points to cluster.
- maxIterations is the maximum number of iterations to run.
- initializationMode specifies either random initialization or initialization via k-means||.
- initializationSteps determines the number of steps in the k-means|| algorithm.
- epsilon determines the distance threshold within which we consider k-means to have converged.
- initialModel is an optional set of cluster centers used for initialization. If this parameter is supplied, only one run is performed.

The KMeans method upon computation returns an array of samples which are to be treated as input samples during training the classifier.

3.2.2 Bisecting K-Means

Bisecting k-means algorithm is a kind of divisive algorithms i.e- it is a “top down” approach: all observations start in one cluster, and splits are performed recursively as one moves down the hierarchy.

The implementation in Spark MLlib has the following parameters:

- k: the desired number of leaf clusters (default: 4). The actual number could be smaller if there are no divisible leaf clusters.
- maxIterations: the max number of k-means iterations to split clusters (default: 20)
- minDivisibleClusterSize: the minimum number of points (if >= 1.0) or the minimum proportion of points (if < 1.0) of a divisible cluster (default: 1)
- seed: a random seed (default: hash value of the class name)

This method too upon completion returns an array of samples which are to be treated as input samples during training the classifier.

3.3 Proposed Scalable Kernel Laplacian PCA based Algorithm

Steps of the modified Algorithm: :

1. Read the input Dataset from HDFS
2. Reduce the dataset using above discussed sampling methods
3. Compute kernel matrix from the reduced sample set
4. Convert the matrix to normalized Laplacian
5. Compute the EVD of Laplacian Matrix
6. Take the projection of the Laplacian on eigen vectors
7. Cluster the data using K means

During the clustering, the Big Data Sampler reduces the number of input samples to a number N such that the kernel matrix of order N can be decomposed using Eigen Value Decomposition. The sampler implementation is such that it works in scalable distributed manner handling Big Data, thereby producing a smaller sample set yet representing the original dataset without losing much of accuracy. The overall training algorithm is designed in a way that it leverages the compute power of cluster through effective use of Spark MLlib APIs.

The proposed algorithm is also implemented in Matlab2017A and exhausitively tested on various cluster validity index.

Chapter 4

Setup & Implementation

This chapter discusses about the hardware & software setup utilized to implement training & testing algorithms on the Apache Spark framework. MATLAB 2017A was used to calculate results for Small Datasets.

4.1 Apache Spark Cluster setup

As discussed in chapter 2, Apache Spark cluster can be deployed in any of the three modes:

1. Standalone
2. Hadoop Yarn
3. Spark in Map Reduce

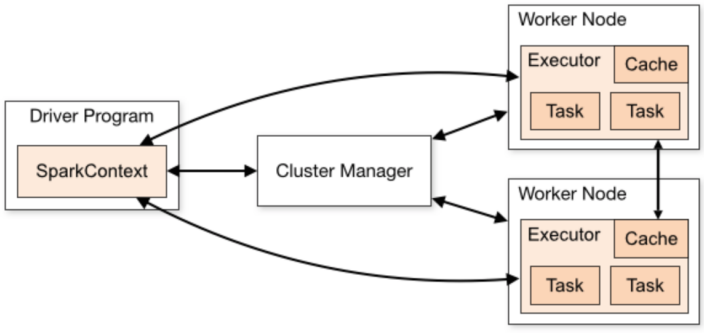


Fig. 4.1 Apache Spark Cluster Overview

For implementing the previously discussed algorithms, the Standalone mode of deployment is used as it is simple to implement and meets all the requirements of this project.

Hadoop Filesystem(HDFS) was installed on all the nodes in the cluster which serves as source to read input dataset and destination for writing the intermediate output.

Spark v2.1 built for Hadoop 2.7 and later was used in the course of this project. The prebuilt version is available to download from the official Apache Spark website. For detailed explanation to get HDFS & Apache Spark up and running on a local cluster, refer the Appendix section.

4.2 Hardware Setup for Cluster Deployment

The standalone cluster was setup using 3 High Performance Computing System. One of the systems doubled up as the Master Node as well as Worker Node while the rest were simply Worker Nodes. All systems run Ubuntu 16.04LTS operating system. The specifications of Master Node and Worker nodes are given below:

Master Node

- 4 cores
- 64GB RAM
- Intel Xeon Processor

Worker Node - 2 Nodes

- 4 cores
- 32GB RAM
- Intel Xeon Processor

Chapter 5

Experimental Analysis & Results

This chapter presents the results obtained by implementing the algorithms discussed in Chapter 3. Both the sampling techniques were applied on all the datasets mentioned in Appendix A. This chapter also discusses the performance gains obtained on scaling up the cluster.

5.1 Clustering Accuracy on various datasets

This section discusses the clustering accuracy achieved by the Kernel Graph PCA application of different sampling methods to reduce the Big dataset.
In all the sampling methods, the Big Dataset was sampled down by reduction ratio 100.

The proposed algorithm is also compared with another variant of graph laplacian which forms a different optimization problem.The experimental result shows that the proposed graph laplacian works better than the variant of Graph Laplacian[5].

5.2 Evaluation Criteria for Clustering

Here we discuss various cluster validity indexes which include NMI,ARI,PURITY and ACCURACY.

Normalized Mutual Index(NMI)

NMI is used to compute the clustering results, which measure the agreement of the clustering results produced.by an algorithm and the ground truth. If we refer to class as the ground truth and to cluster as the results of a clustering algorithm, the NMI is calculated as follows:
Where, n is the total number of data points, n_c and n_p are the numbers of data-points in the c

$$NMI = \frac{\sum_{c=1}^{k}\sum_{p=1}^{m} n_c^p \log\left(\frac{n.n_c^p}{n_c.n_p}\right)}{\sqrt{\left(\sum_{c=1}^{k} n_c \log\left(\frac{n_c}{n}\right)\right)\left(\sum_{p=1}^{m} n_p \log\left(\frac{n_p}{n}\right)\right)}}.$$

th cluster and the p th class, respectively, and n_c^p is the number of common data-points in class p and cluster c

Adjusted Random Index(ARI)

The adjusted Rand index is the corrected for chance version of the Rand index, which is a measure of the similarity between two data clustering. To compute the ARI, we first harden the fuzzy partitions by setting the maximum element in each column of U to 1, and all else to 0. We use ARI to compare the clustering solutions with ground-truth labels (when available), as well as to compare other algorithms with the Spectral Clustering. The formulation of ARI is defined as follows::

$$ARI = \frac{\sum_{i,j}\binom{n_{ij}}{2} - \left[\sum_i \binom{n_{i.}}{2}\sum_j \binom{n_{.j}}{2}\right]/\binom{n}{2}}{\frac{1}{2}\left[\sum_i \binom{n_{i.}}{2} + \sum_j \binom{n_{.j}}{2}\right] - \left[\sum_i \binom{n_{i.}}{2}\sum_j \binom{n_{.j}}{2}\right]/\binom{n}{2}}.$$

PURITY

All the datasets we used have labels. We view the labels of the datasets as the objective knowledge on the structure of the datasets. We use purity as the clustering performance measure. purity discovers the one-to-one relationship between clusters and classes and measures the extent to which each cluster contains data points from the corresponding class and it has been used as performance measures for clustering analysis. Purity can be described as:

$$Max\left(\sum_{C_k, L_m} T(C_k, L_m)\right)/n,$$

where n is the number of data points, C_k denotes the k -th cluster, and L_m is the m -th class. T (C_k , L_m) is the number of data points that belong to class m are assigned to clus-

ter k . Purity is then computed as the maximum sum of T (C_k , L_m) for all pairs of clusters and classes, and these pairs have no overlaps.

ACCURACY

The ACCURACY(AC) is defined as follows:

$$AC = \frac{\sum_{i=1}^{n} \delta(s_i, map(r_i))}{n}$$

where n is the total number of samples and

delta(x, y) is the delta function that equals one if x = y and equals zero otherwise, and map(r_i) is the permutation mapping function that maps each cluster label r_i to the best label from the data set. The best mapping can be found by us ing the Kuhn-Munkres algorithm

The results depict that even with large reduction ratios in case of Replicated IRIS and Replicated Musk, the clustering accuracy is pretty reasonable. Due to limit on matrix size order for eigenvalue computation, large sample sizes couldn't be tested. Theoretically, increased number of samples should improve the clustering accuracy.But this is not always true. This depends on vector quantization based sampling i.e the representative data points of large sets.

5.3 Computational Speedups and Experimental Results for Big Data

This section presents the performance gains obtained on scaling the cluster in terms of number of compute units. The results represented are for varying core counts for datasets of varying sizes.

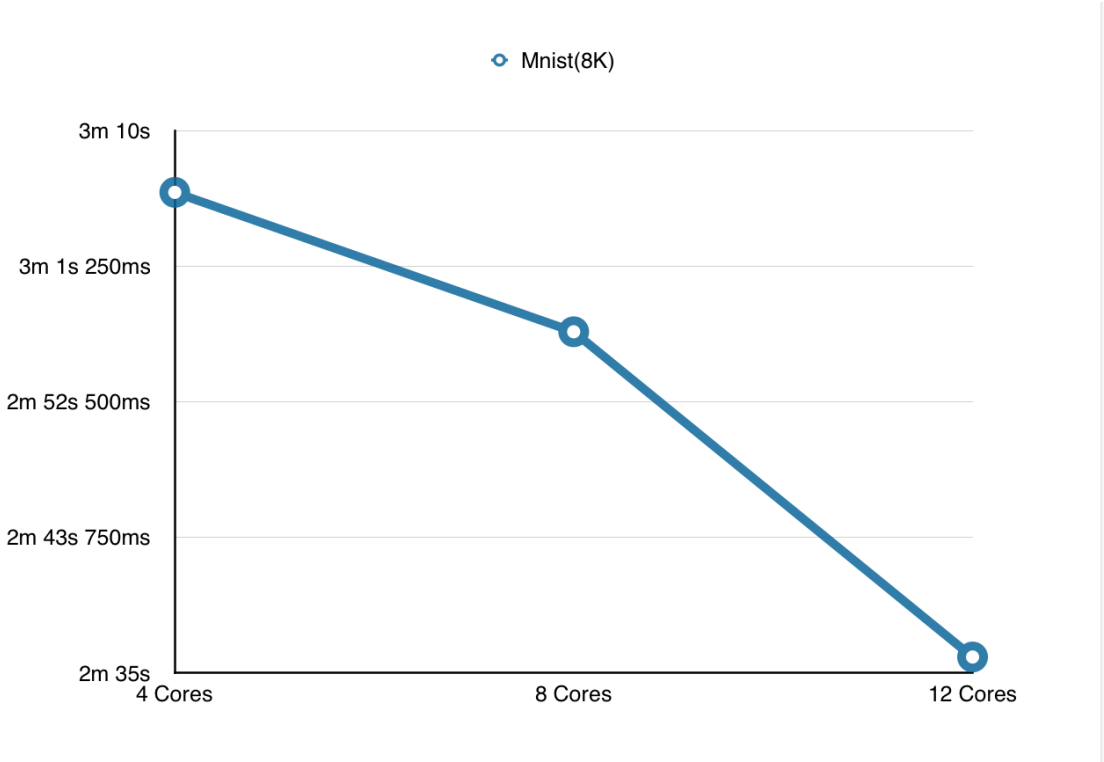


Fig. 5.1 Gaussian Kernel for size 8K

Table 5.1 Dataset: Replicated IRIS(150K) for Kmeans

Clustering Method	NMI	TIME
K-Means	0.7586	0.08333
Bisecting K-Means	0.7841	0.05327

Table 5.2 Dataset: Replicated IRIS(150K) for Spectral Clustering

Sampling Method	NMI	TIME
K-Means	0.8642	18.998
Bisecting K-Means	0.8981	8.8447

Table 5.3 Dataset: Replicated IRIS(150K) for Kernel PCA

Sampling Method	NMI	TIME
K-Means	0.00104	62.66
Bisecting K-Means	0.0456	78.7

Table 5.4 Dataset: Replicated IRIS(150K) for Kernel Graph PCA

Sampling Method	NMI	TIME
K-Means	**0.8642**	**18.998**
Bisecting K-Means	**0.8981**	**8.8447**

Table 5.5 Dataset: Replicated MUSK(150K)

Clustering Method	NMI	TIME
K-Means	0.0542	10.837
Bisecting K-Means	0.0613	8.324

Table 5.6 Dataset: Replicated MUSK(150K) for Spectral Clustering

Sampling Method	NMI	TIME
K-Means	0.0384	104.34
Bisecting K-Means	0.0395	118.23

Table 5.7 Dataset: Replicated MUSK(150K) for Kernel PCA

Sampling Method	NMI	TIME
K-Means	0.0473	98.543
Bisecting K-Means	0.0453	100.07

Table 5.8 Dataset: Replicated MUSK(150K) for Kernel Graph PCA

Sampling Method	NMI	TIME
K-Means	**0.0626**	**106.567**
Bisecting K-Means	**0.0733**	**120.674**

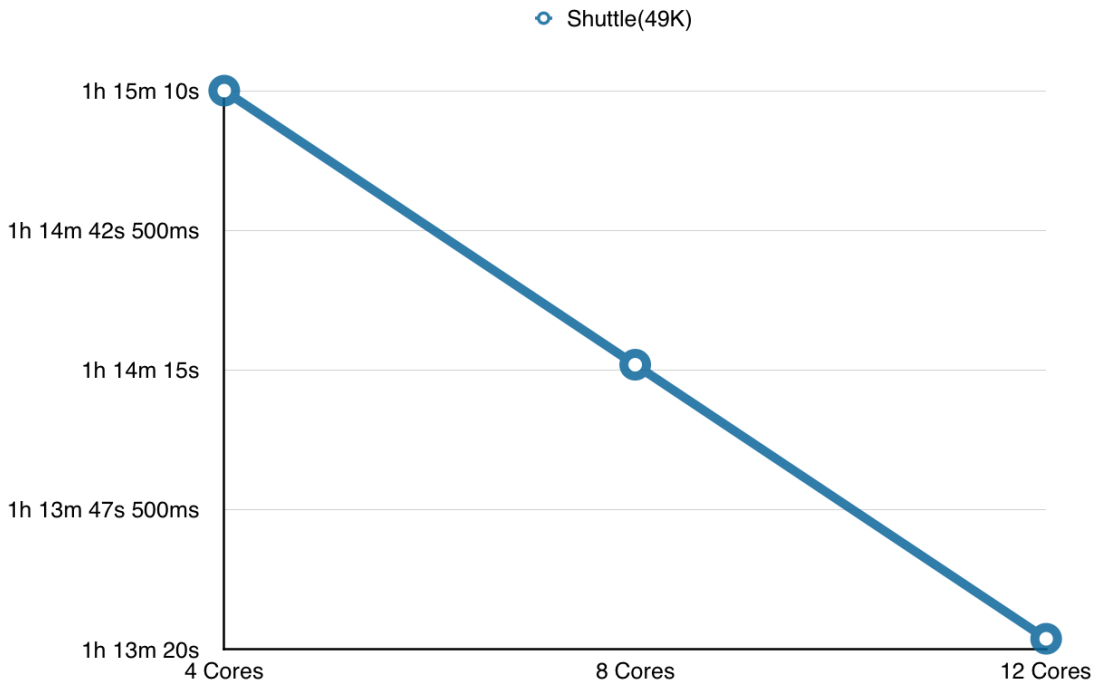


Fig. 5.2 Gaussian Kernel for size 49K

The above graphs clearly depict the scalability of the algorithms. Increased core counts is reflected in reduced compute times. The algorithm leverages the ability of Apache Spark framework to increase performance with upscaling of cluster.

5.4 Experimental Results for Small DataSets

In the experiments, we compare the performance of Kernel PCA ,Kmeans+PCA, Kernel Graph PCA , LDA-KM and Graph Laplacian using various measures.All the approaches are implemented on an Apache Spark Cluster

	SPECTRAL CLUSTERING	KMEAN +PCA	KERNEL PCA	LDA -KM	KERNEL GRAPH PCA
ZOO	0.883	0.702	0.783	0.842	**0.921**
IONOSPHERE	0.758	0.71	0.768	0.712	**0.784**
IRIS	0.892	0.887	0.902	0.985	**0.947**
WINE	0.754	0.702	0.742	0.826	**0.916**
GLASS	0.543	0.453	0.532	0.547	**0.589**
DIGITS	0.792	0.768	0.765	0.772	**0.821**
SOYBEAN	0.841	0.723	0.851	0.768	**0.894**

Fig. 5.3 Results based on PURITY measure

DATASET	EVALUATION MEASURE	1	2	3	4	5	BEST ACCURACY
ISOLET	ARI	0.6178	0.6194	0.6174	0.6179	0.6661	**0.6317**
	NMI	0.779	0.7821	0.7791	0.7807	0.7824	**0.8038**
	PURITY	0.7397	0.7494	0.7327	0.7335	0.7498	**0.7776**
	ACCURACY	0.6324	0.6327	0.6311	0.6529	0.6544	**0.6750**
LIBRAS	ARI	0.3440	0.3464	0.3413	0.3437	0.3375	**0.3546**
	NMI	0.6221	0.6252	0.6206	0.6203	0.6233	**0.6351**
	PURITY	0.6178	0.6000	0.5962	0.6028	0.5917	**0.6000**
	ACCURACY	0.475	0.4806	0.4889	0.4891	0.4917	**0.4889**
MFEAT	ARI	0.5135	0.4331	0.5429	0.5416	0.5438	**0.5967**
	NMI	0.7010	0.6947	0.7248	0.6477	0.6493	**0.7258**
	PURITY	0.8000	0.7500	0.7605	0.7580	0.7590	**0.7875**
	ACCURACY	0.6460	0.5615	0.6540	0.6700	0.669	**0.6720**

Fig. 5.4 Results for Kernel Graph PCA on various evaluation measures
Average accuracy has been reported by increasing the multiplicity of k where k is number of clusters and running the algorithm for 50 times. The best accuracy is reported corresponding to the highest average accuracy of the multiple of k by taking the best results when run for 50 times

DATASET	EVAL MEASURE		KMEAN +PCA	KPCA	SC	KGPCA	GRAPH LAPLACIAN
ISOLET1	ARI	AVG	0.5467	0.5789	0.5401	**0.6661**	
		BEST	0.5687	0.5980	0.5459	**0.6717**	
	NMI	AVG	0.7248	0.7611	0.7696	**0.7824**	0.7669
		BEST	0.7367	0.7811	0.7845	**0.8038**	
	PURITY	AVG	0.6295	0.7282	0.7417	**0.7498**	0.6638
		BEST	0.6487	0.7342	0.7577	**0.7776**	
	ACC	AVG	0.5860	0.6090	0.6033	**0.6544**	0.6431
		BEST	0.5960	0.6255	0.6295	**0.6750**	
LIBRAS	ARI	AVG	0.3122	0.2771	0.3092	**0.3464**	
		BEST	0.3243	0.2992	0.3350	**0.3546**	
	NMI	AVG	0.583	0.5674	0.5912	**0.6252**	0.6178
		BEST	0.612	0.5833	0.5957	**0.6351**	
	PURITY	AVG	0.597	0.5167	0.5667	**0.6178**	0.5052
		BEST	0.5863	0.5583	0.5389	**0.6248**	
	ACC	AVG	0.4454	0.4139	0.4583	**0.4917**	0.4687
		BEST	0.4657	0.4278	0.5000	**0.5123**	
MFEA	ARI	AVG	0.4937	0.4837	0.4927	**0.5438**	
		BEST	0.5328	0.5428	0.5538	**0.5967**	
	NMI	AVG	0.6303	0.7003	0.7008	**0.7248**	0.7043

Fig. 5.5 Results
The average accuracy is the average of results over 50 runs. The best accuracy is the best of results over 50 runs.

MFEA	NMI	BEST	0.6424	0.7124	0.7142	**0.7258**	
	PURITY	AVG	0.6776	0.6876	0.6908	**0.8000**	0.6899
		BEST	0.7125	0.7425	0.7467	**0.8125**	
	ACC	AVG	0.6327	0.6387	0.6402	**0.6700**	0.6472
		BEST	0.6457	0.6527	0.6534	**0.6720**	

Fig. 5.6 Extended results

Chapter 6

Conclusion and Future Scope

Kernel Based Algorithm is a promising straightforward algorithm which can achieve a relatively high Overall Accuracy on Clustering. However, massive data or Big Data is a big challenge for the practical use of Kernel Clustering in terms of both space complexity and computational time complexity. Through this project we implemented a novel Kernel Graph PCA and obtained significant levels of computational speedup when scaling up the cluster having more accuracy then KPCA and Spectral Clustering. Through this project, we realized the advantages, capabilities and limitations of Apache Spark framework and gained significant insights in the fields of Machine Learning and Big Data Handling.

While implementing the algorithm,we hit a roadblock where we got restricted on the size of the Kernel matrix. For future research, perhaps a better distributed way to compute kernel matrix Eigen Value Decomposition can be developed and implemented which would improve the overall accuracy of clustering.Also Sampling techniques can be further enhanced to handle big data. Nystrom sampling technique can be used.

References

[1] Bharill, N., Tiwari, A., and Malviya, A. (2016). Fuzzy based scalable clustering algorithms for handling big data using apache spark. *IEEE Transactions on Big Data*, 2(4):339–352.

[2] Dhillon, I. S., Guan, Y., and Kulis, B. (2004). Kernel k-means: spectral clustering and normalized cuts. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 551–556. ACM.

[3] Ding, C. and Li, T. (2007). Adaptive dimension reduction using discriminant analysis and k-means clustering. In *Proceedings of the 24th international conference on Machine learning*, pages 521–528. ACM.

[4] Hofmann, T., Schölkopf, B., and Smola, A. J. (2008). Kernel methods in machine learning. *The annals of statistics*, pages 1171–1220.

[5] Jiang, B., Ding, C., and Tang, J. (2013). Graph-laplacian pca: Closed-form solution and robustness. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3492–3498.

[6] Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., et al. (2016). Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17(1):1235–1241.

[7] Ng, A. Y., Jordan, M. I., and Weiss, Y. (2002). On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pages 849–856.

[8] Rekha, A. (2015). A fast support vector data description system for anomaly detection using big data. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 931–932. ACM.

[9] Spark, A. Bisecting k-means clustering.

[10] Spark, A. K-means clustering.

[11] Vert, J.-P., Tsuda, K., and Schölkopf, B. (2004). A primer on kernel methods. *Kernel Methods in Computational Biology*, pages 35–70.

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

[13] Wang, H., Hu, Z., and Zhao, Y. (2006). Kernel principal component analysis for large scale data set. *Lecture Notes in Computer Science*, 4113:745.

[14] Wang, S., Gittens, A., and Mahoney, M. W. (2017). Scalable kernel k-means clustering with nystrom approximation: Relative-error bounds. *CoRR*, abs/1706.02803.

Appendix A

Datasets used for Clustering

A description of the datasets used for clustering purposes.

Shuttle Dataset

The original Statlog (Shuttle) dataset from UCI machine learning repository is a multi-class classification dataset with dimensionality 9.

Mnist Dataset

The original MNIST dataset of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

ISOLET Dataset

This data set was generated as follows. 150 subjects spoke the name of each letter of the alphabet twice. Hence, we have 52 training examples from each speaker. The speakers are grouped into sets of 30 speakers each, and are referred to as isolet1, isolet2, isolet3, isolet4, and isolet5. The data appears in isolet1+2+3+4.data in sequential order, first the speakers from isolet1, then isolet2, and so on. The test set, isolet5, is a separate file.We used isolet1 for the algorithm.

MultiFeatures Dataset

This dataset consists of features of handwritten numerals (‘0’–‘9’) extracted from a collection of Dutch utility maps. 200 patterns per class (for a total of 2,000 patterns) have been digitized

in binary images.These digits are represented in terms of the six feature sets (files): We used Multi Features Fac dataset.

COIL 20 Dataset

Columbia Object Image Library (COIL-20)is a database of gray-scale images of 20 objects.The objects were placed on a motorized turntable against a black background.The turntable was rotated through 360degrees to vary object pose with respect to a xed camera.Images of the objects were taken at pose intervals of 5degrees.This corresponds to 72images per object.The database has two sets of images.The rest set contains 720 unprocessed images of 10 objects.The second contains 1,440 size normalized images of 20 objects.

Other Datasets from UCI

IRIS , Pen digits, Libras , Ionosphere , Glass and Soyabean were also used.

Appendix B

Apache Spark and Hadoop - Setup Guidelines

B.1 Setting up Environment

B.1.1 Installing Java & Scala

In order to get Hadoop and Apache Spark running on your systems, Java(Java 8) and Scala(2.12 and above) are required. This section provided instruction to install the same.

Installing Java

```
1. Open terminal
2. sudo apt-get update
3. sudo apt-get install openjdk-8-jdk
```

Installing Scala

```
1. Open terminal
2. sudo apt-get update
3. sudo apt-get install scala
```

B.1.2 Setting up password-less SSH

All the systems in the cluster need to communicate with each other seamlessly in order to start and stop processes and services.Hence setting up of password-less SSH is essential. The link provided below explains all the steps to do the same.

http://www.linuxproblem.org/art_9.html

B.2 Setting up Hadoop on Ubuntu

With Java installed on your system, the below linked URL provides detailed guidelines to install Hadoop(Hadoop 2.7.3) on each system of your cluster Since Java is already installed on the system, proceed directly to Step 2 in the tutorial.

Link:- https://www.digitalocean.com/community/tutorials/how-to-install-hadoop-in-stand-alone-mode-on-ubuntu-16-04

Once Hadoop is installed on each system of the cluster, follow the steps in the URL given below to setup of Hadoop cluster.

Link:- http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-multi-node-cluster/

B.3 Setting up Apache Spark

The prebuilt version of Apache Spark is available on the offical Apache Spark website.The link for the same is given below. This prebuilt version needs to be downloaded and extracted on each system which is part of the cluster.

Link:- https://archive.apache.org/dist/spark/spark-2.1.0/spark-2.1.0-bin-hadoop2.7.tgz

Once Apache Spark spark has been downloaded and installed on each system, follow the steps in tutorial on the link below to setup Spark Standalone Cluster:

Link:- http://paxcel.net/blog/how-to-setup-apache-spark-standalone-cluster-on-multiple-machine/

B.4 Running Spark Jobs on Cluster

In order to run spark jobs written in scala follow the steps given below:

```
1.On master system,open terminal
2.cd <path-to-prebuilt-spark>
3. ./bin/spark-shell --master spark://<master IP>:7077 --total-executor-cores <no. of
4. To compile the code -> :load <name of file>.scala
5. To run the code -> <name of object>.main(Array(<arguments>))
```