B.Sc. in Computer Science and Engineering Thesis

Geo Distributed Implementation of PPCA on Tall and Wide Big Data

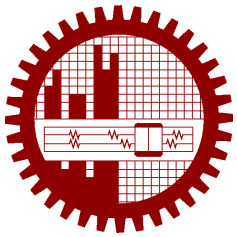
Submitted by

Md. Rashedul Islam
201205013

T.M. Tariq Adnan
201205073

Supervised by

Dr. Muhammad Abdullah Adnan



Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology

Dhaka, Bangladesh

February 2017

CANDIDATES' DECLARATION

This is to certify that the work presented in this thesis, titled, "Geo Distributed Implementation of PPCA on Tall and Wide Big Data", is the outcome of the investigation and research carried out by us under the supervision of Dr. Muhammad Abdullah Adnan.

It is also declared that neither this thesis nor any part thereof has been submitted anywhere else for the award of any degree, diploma or other qualifications.

Md. Rashedul Islam
201205013

T.M. Tariq Adnan
201205073

CERTIFICATION

This thesis titled, **"Geo Distributed Implementation of PPCA on Tall and Wide Big Data"**, submitted by the group as mentioned below has been accepted as satisfactory in partial fulfillment of the requirements for the degree B.Sc. in Computer Science and Engineering in February 2017.

Group Members:

Md. Rashedul Islam

T.M. Tariq Adnan

Supervisor:

Dr. Muhammad Abdullah Adnan
Assistant Professor
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology

ACKNOWLEDGEMENT

First of all, we would like to express our gratitude to our supervisor Assistant Professor Dr. Muhammad Abdullah Adnan for introducing us to the large world of Machine Learning and Big Data and for showing us the appropriate approach to science. We thank him for his patience in times of our countless missteps, support in our work, excellent advices and constant supervision. Especially, his encouragement in our tough days was invaluable.

We also express our gratitude to Md. Mehrab Tanjim to help us in enormous ways. Without him it would not be possible to do the hard works.

We would also like to express our utmost gratitude to Professor Dr. M. Sohel Rahman, Head, Department of Computer Science and Engineering, BUET, for the provision of laboratory facilities.

Finally, we would like to acknowledge the help and support of members of our research group.

Dhaka
February 2017

Md. Rashedul Islam

T.M. Tariq Adnan

Contents

List of Figures

List of Tables

List of Algorithms

ABSTRACT

Nowadays, there are restrictions on data to cross national boundaries to preserve privacy and uphold national security. However, to ensure quick access, data centers are spread across multiple nations in order to reside data as close to the end users as possible. On top of this, rapid growth and ubiquitous generation of data are causing big data becoming both tall and wide. In these new settings, analytic techniques like *PCA* are seeing a new set of challenges. In terms of computational and communicational complexities, Probabilistic Principal Component Analysis (PPCA) holds significantly higher efficiency. It also has some major advantages over other techniques such as set up a probabilistic model and ability to handle missing data which is highly advantageous in case of geographically sparse data. Unfortunately currently we posses no suitable method to calculate *PPCA* on geographically distributed big data which is both tall and wide. In this paper, we propose a new approach to handle tall and wide big data in case of performing *PPCA* in a single data cluster (or a simple commodity computer) and accumulate the intermediate results to generate the final one. Moreover we provide an implementation for it. We hope our work has a great potential in popularizing *PPCA* as an practical analytical tool for big data by eliminating the dimensional and geographical restrictions and thus holds a significant promise in this premise.

Chapter 1

Introduction

The frequency of data access at the users' end has been increased by a large number for the past few years. To ensure low latency at the users' end, it is preferable to reside the data as close to the users as possible. Moreover, in present world data privacy has been a great concern and almost all the nations require the data of their citizens not to reside in some place across their national border. Now its a big concern to maintain privacy regulations and keep foreign countries from being able to subpoena data. To mitigate these two issues, almost all the companies have been building their data centers all around the world. As a result, the sparsity of data has been increased by a huge amount during the last few years.

On the other hand, the amount of data generating in present world is quite large. This huge volume has introduced a new term "Big Data". In general "Big Data" is nothing but the large volume of data that can be both structured and unstructured. The excessive use of social media, scientific instruments, portable devices, sensors are the main reason behind the generation of big data. Data analysis, capture, curation, search, share, storage, transfer, visualization, query, updating, information privacy everything associated with big data has been the challenges of the present world. In case of big data the can be both wide and tall that means the dimension of the data can be quite large. As a result available techniques might not be able to perform the data analysis with the provided limited hardware. Therefore, we need approaches that have the property of scalability.

For researchers, however, big data brings new sets of challenges like how to efficiently and effectively handle big data in commodity computers, how to apply conventional algorithms, how to properly manage big data in distributed setting etc. Specially, in field of Machine Leaning and Pattern Recognition, big data poses a threat, because the conventional algorithms were designed solely to fulfil its purpose where entire dataset can fit in memory. Distributed setting was not taken into consideration. In a nutshell, to extract any meaningful insight from this big data, we need powerful tools to analyse it, elastic frameworks to cope up with its sheer volume and most importantly we have to rethink and redesign existing algorithms which are most suitable for

smaller sized dataset and do not take advantage of clusters. In this thesis, we consider one kind of the machine learning algorithms, namely Probabilistic Principal Component Analysis(PPCA) and try to make it practically usable in analysing tall and wide big data which are distributed throughout the whole world. In this chapter we provide the motivation behind our research in Section 1.1, we discuss the advantages of using PPCA in Big Data Analysis in Section 1.2, in Section 1.3 we present our contribution in data analysis field and in Section 1.4 we briefly describe the organization of this book.

1.1 Motivation

Let us assume some scenario. Let us assume that We have good availability of machine learning models to do various kinds of data analytics. Moreoevr, our data is of the size 10000×200. That means it contains 10000 sample which each sample has 200 features. The machine learning models will be capable to fit such data pretty easily. As soon as the data usage and data generation rate increases the data size gets larger to 200000×2000. This time the machine learning models fails to fit such higher dimensional data. Fortunately, we have Principal Component Analysis (PCA) technique to reduce the dimensionality of the data. Thus we can get 100 principal components from the data. Now the data is compressed enough to fit in our Machine learning models. But if the dimension of data become more higher like in millions or billions then such Tall and Wide data will neither be suitable to fit in machine learning models nor the tradition PCA algorithms can handle such large dimensional data. More question arise if the data get to be geographically distributed. We were intended to give such a method that can answer all these questions and hence solve the problems of geographically distributed higher dimensional Tall and Wide Data.

1.2 Probabilistic PCA (PPCA) on Big Data

As PPCA incorporates both probabilistic model of data and EM algorithm for deriving the parameters, it has some advantages from from both sides. Here we elaborate the major ones:

- Firstly and most importantly, EM algorithm for deriving the parameter is computationally efficient, if only few principal components are required. If we want only k components, where $k << D$ and size of data is $N \times D$, time complexity of PPCA is $O(NDk)$. It does not require to calculate a big dense data covariance matrix as intermediate data, so communication complexity is also less. Calculating data covariance matrix is costly in terms of both computational and communication complexity. For this reason. deterministic PCA methods (like EVD, SVD) do not scale well with data (time complexity: $O(ND^2)$).

More detailed analysis can be found in the technical report [32]. Thus EM steps in PPCA give a well scalable algorithm and is a great advantage for big data analysis.

- For large dataset, it is very common that some entries of data can be missing. As big data can have missing values, deterministic PCA algorithm will give erroneous result and will not be suitable. But for PPCA it is not a problem. Since iterative method of PPCA uses expectation maximization, projected data or expected values of latent variable can still be derived even if some values are missing in random. This benefit of handling missing values is one of the major advantages of PPCA.

- Deterministic PCA methods are not aware of the isotropic noise that is considered by PPCA. It is because these methods consider the squared distance of the new data points from their projections into the principal subspace. So, even for data points that are randomly far from the training data, conventional PCA will give a low reconstruction cost if these points are close to the principal subspace. We do not face similar problems in PPCA as it takes noise as a parameter and assigns likelihood accordingly.

- For a well defined problem, target dimension k can be known beforehand. But in many data reduction problems the optimal It may not be known in advance. Thus we need such a method that can find dimensionality of the principal subspace automatically from the data. This can be done by a Bayesian treatment of PCA and PPCA forms the basis for it. This is a great advantage for unsupervised learning.

- As we are deriving parameters of a probabilistic model of data, we can easily generate 'synthesized' or 'fantasy data from the distribution.

- We can combine multiple PPCA models as a probabilistic mixture for complex models to get better accuracy. Using EM algorithm we can easily train mixtures of PPCA models.

- Number of independent parameters in PPCA model grows only linearly with dimension of data D for a fixed target dimension k. Because of this reason, number of independent parameters for multivariate Gaussian distribution can be restricted while still allowing if to capture the dominant correlation in data.

- Lastly, PPCA can be applied to classification problems because we can use it to model class-conditional densities.

1.3 Contribution

Summarizing, our main contributions are:

- We propose an approach of PPCA that deals with geographically distributed Tall and Wide Big Data and provide an in-depth study of the relative merits against recent efficient approach *sPCA* [22].
- We propose a system that builds upon Apache Spark [35] environment and extends their functionality to support multiple data clusters learning applications.
- We propose an apparently time efficient method to accumulate partially generated results in various data clusters.
- Our proposed approach will minimize the number of transfers among data centers thus minimize the inter data center bandwidth utilization.
- Our method will be scalable to the available hardware of the data clusters. It will allow us to handle Tall and Wide big data in commodity computers even with memory of $6/8GB$.
- We will present our experimental results implemented on the clusters in our undergraduate thesis laboratory that will justify the validity of our method.

1.4 Organization of this Book

The rest of the book is organized as follows. In Chapter 2, we define some basic terminologies. In Chapter 3, we describe all the mathematical background characteristics of PCA. In Chapter 4, we write about our focus. In Chapter 5, we describe our preliminary works on regression analysis. In Chapter 6, we describe our main approach to solve the existing problem. Chapter 7 describes the properties of our approach while Chapter 8 gives a brief on our Spark implementation. With showing the experimenta;l results in Chapter 9, the book concludes with Chapter 10.

Chapter 2

Basic Terminologies

2.1 Big Data

Big Data is a term for voluminous amounts of data that has the potential to be mined for information. Big data can be analyzed for insights that lead to better decisions. Big Data is characterized by three main factors: the extremely large volume of data, the wide variety of data types and the velocity at which the data must be processed.

Such voluminous data can come from countless different sources such as business sales records, the collected results of scientific experiments of real-time sensors used in the internet of things (IoT).

Data may exist in wide variety of types, including structured data, such as SQL database stores, as well as unstructured data such as document files or streaming data from sensors. Big data of different types and from different sources may be used together during analysis.

Principal Component Analysis(PCA), which is the main focus of our thesis, is considered as a pre-processing step in Big Data Analytic. It reveals the relations between the different dimensions of data and allows a reduction in dimensionality of the data via low rank approximation.

2.2 Data Analysis

According to [1] Data analysis, also known as analysis of data or data analytics, is a process of inspecting, cleansing, transforming, and modeling data with the goal of discovering useful information, suggesting conclusions, and supporting decision-making. Data analysis has multiple facets and approaches, encompassing diverse techniques under a variety of names, in different business, science, and social science domains.

Data mining is a particular data analysis technique that focuses on modeling and knowledge discovery for predictive rather than purely descriptive purposes, while business intelligence covers data analysis that relies heavily on aggregation, focusing on business information [2]. In statistical applications data analysis can be divided into descriptive statistics, exploratory data analysis (EDA), and confirmatory data analysis (CDA). EDA focuses on discovering new features in the data and CDA on confirming or falsifying existing hypotheses. Predictive analytics focuses on application of statistical models for predictive forecasting or classification, while text analytics applies statistical, linguistic, and structural techniques to extract and classify information from textual sources, a species of unstructured data. All are varieties of data analysis.

2.3 Data Analysis Approaches

Under these circumstances, we can see that in case of extracting some information from the existing data, we may have to cover a good number of data centres located at different geographical area. This has given the introduction of a new data analysis approach, "Distributed Data Analysis". Therefore we get the idea of two approaches of data analysis.

2.3.1 Centralized Approach

The centralized approach to data analysis from distributed data centers is to centralize them first. As shown in Figure 2.1, this involves a two different steps:

1. **Centralizing step** Data from various data centers are copied into a single data center. It involves recreation of data of all data centers in one location.
2. **Analysis Step** Process of extracting the necessary information from the centralized data takes place in that single data center. Here traditional intra data center technology is sufficient for the analysis purpose.

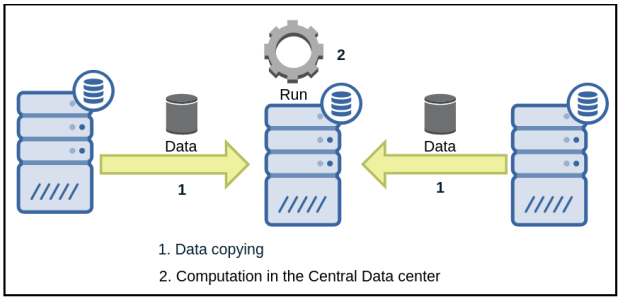


Figure 2.1: Centralized Approach

This centralized approach is predominant in most practical settings. There are mainly two reasons behind its popularity.

1. There are lots of frameworks that have already been established for centralized learning approach. That is why centralizing the data is the easiest way to reuse existing data analysis frameworks [8–10]
2. Learning algorithms are highly communication intensive. Thus it is assumed that they will note be properly responsive to cross data center execution.

For these reasons, this centralized approach is consistent with reports on the infrastructures of other large organizations, such as Facebook [11], Twitter [12], and LinkedIn [13].

However the centralized approach has two shortcomings.

1. While making multiple copy of data at the central data center, it consumes a good amounts of inter data center bandwidth. Since inter data center bandwidth is expensive, it is not easy to increase it according to the necessity [14–17].
2. While creating copy of data, it may be a case that data is crossing national borders. However, in current workd data sovereignty is a growing concern that might create a big limitation in this aspect [18, 19].

2.3.2 Distributed Approach

In the distributed approach, raw data is kept in their corresponding data centers. Every data center does a portion of the execution that is only on the data of that data center. The final analysis takes pales by passing small amount of information among the data centers.

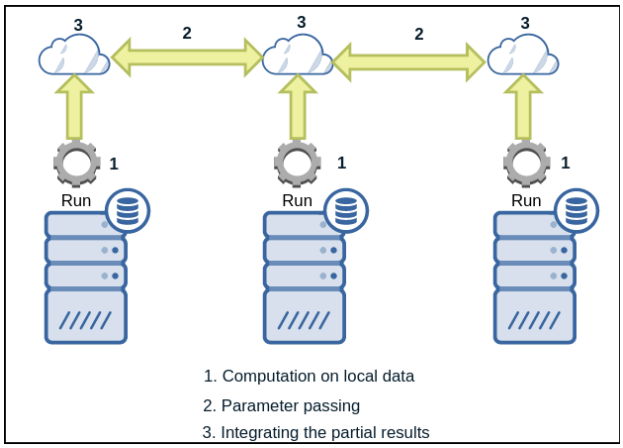


Figure 2.2: Distributed Approach

So according to Figure 2.2, we can see this approach includes three steps:

1. **Local Computation** Whenever the command of starting of any learning process is issued every data center start a partial computation on its own data. They create necessary high level information on data that will be needed in the final computation.

2. **Parameter Passing** Data centers communicate among themselves and share valuable information.
3. **Final Computation** By integrating the partial results data centers make the final computational model.

In this way distributed solutions can achieve much lower cross data centers bandwidth utilization, and thus substantially lower cost for large-scale analysis tasks. As the current world has got a concern about 'Big Data' and 'Data Sovereignty', the distributed approach seems to be more efficient.

2.4 Geo Distributed Data

Over the year concept about data has changed a lot. Only few years ago data were generated locally and computation was done locally also. But now data are generated over the whole world. Data are not bound to locality.

Data are by born geographically distributed over the world. That's why nowadays geo distributed analysis is being popular. Data are not gather in one place rather algorithms are being designed to run over the geo-distributed data.

2.5 Data Sovereignty

Data sovereignty is the concept that information which has been converted and stored in binary digital form is subject to the laws of the country in which it is located [3].

Many of the current concerns that surround data sovereignty relate to enforcing privacy regulations and preventing data that is stored in a foreign country from being subpoenaed by the host country's government.

The wide-spread adoption of cloud computing services, as well as new approaches to data storage including object storage, have broken down traditional geopolitical barriers more than ever before. In response, many countries have regulated new compliance requirements by amending their current laws or enacting new legislation that requires customer data to be kept within the country the customer resides.

Verifying that data exists only at allowed locations can be difficult. It requires the cloud customer to trust that their cloud provider is completely honest and open about where their servers are hosted and adhere strictly to service level agreements (SLAs).

2.6 Data Cluster and Cluster Computing

A cluster is a system comprising two or more computers or systems (called nodes) which work together to execute a particular task. They are usually deployed for High Availability (HA) for greater reliability and High performance Computing (HPC) to provide greater computational power than a single computer can provide.

The components of a cluster are usually connected to each other through fast local area networks (LANs), with each node running its own instance of an operating system(OS). In most circumstance, all of the nodes use the same hardware and the same OS, although it is possible to user different OS and/or different hardware on each computer.

2.6.1 Advantages

Cluster computing provides the following important advantages.

1. Cluster computing is completely a scalable solution. Resources may be removed or new resources may be added to clusters after the system has already been deployed. They can scale to very large and complex systems with large computing power, not possible with single computers. Also each of the machines in a cluster can be a complete system, usable for a Wide range of other computing applications.
2. Clustering solutions are more fault tolerant. If one of the servers in the system stops working, other servers in the cluster can take the load. If a server in the cluster needs any maintenance, it can be done by stopping it while handing the load over to other servers.
3. Clusters are more cost effective. A cluster will cost much less than a single computer of comparable speed and availability.

2.6.2 Disadvantages

Unfortunately, clustering suffers from some limitations as well:

1. As clusters consist of many computers running in parallel, it is obvious that these systems are only efficient in carrying out a specific task if the task can be separated into smaller subtasks that can be completed in parallel. This may not always be possible.
2. The network hardware used to communicate typically has low bandwidth and high latency (as compared to the link between the different processors in a single computer) and this communication complexity usually acts as the bottleneck during execution of a task.

Thus, algorithms are required to be designed to run on distributed settings and must be optimized to reduce communication between nodes which may be tough to do.

3. Clusters can become very large and complex, and the maintenance of these large and complex systems could be quite expensive.

2.7 Data Parallelism

Data parallelism is a form of parallelization across multiple processors in parallel computing environments. It focuses on distributing the data across different nodes, which operate on the data in parallel. It can be applied on regular data structures like arrays and matrices by working on each element in parallel. It contrasts to task parallelism as another form of parallelism [4].

A data parallel job on an array of 'n' elements can be divided equally among all the processors. Let us assume we want to sum all the elements of the given array and the time for a single addition operation is Ta time units. In the case of sequential execution, the time taken by the process will be $n * Ta$ time units as it sums up all the elements of an array. On the other hand, if we execute this job as a data parallel job on 4 processors the time taken would reduce to $(n/4) * Ta$ + Merging overhead time units. Parallel execution results in a speed up of 4 over sequential execution. One important thing to note is that the locality of data references plays an important part in evaluating the performance of a data parallel programming model. Locality of data depends on the memory accesses performed by the program as well as the size of the cache.

2.8 Model Parallelism

Data Parallelism and Model Parallelism are different ways of distributing an algorithm. These are often used in the context of machine learning algorithms that use stochastic gradient descent to learn some model parameters [6].

In model parallelism algorithm sends the same data to all the cores. Each core is responsible for estimating different parameter(s). Cores then exchange their estimate(s) with each other to come up with the right estimate for all the parameters.

2.9 TensorFlow

TensorFlow [7] is an open source software library for numerical computation using data flow graphs. As we have used the concept of model parallelism and data parallelism in our geo

distributed algorithm, so it is important. Because in *TensorFlow* architecture both the model parallelism and data parallelism have been used.

The flexible architecture of *TensorFlow* allows to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API. *TensorFlow* was originally developed by researchers and engineers working on the *Google* Brain Team within *Google*'s Machine Intelligence research organization for the purpose os conducting machine learning and deep neural networks research, but the system is general enough to be applicable in wide variety of other domains as well.

2.10 *Hadoop* Cluster

Hadoop [37] is an open source software framework built on two technologies, Linux operation system and Java programming language. It supports the processing and storage of extremely large data sets in a cluster computing environment. It is part of the Apache project sponsored by the Apache Software Foundation.

All the modules in *Hadoop* are designed with the assumption that hardware failures are common occurrences and should be automatically handled by the framework.

2.11 *Hadoop* MapReduce

MapReduce is the heart of *Hadoop*. MapReduce is a processing technique and a programming paradigm for distributed computing based on Java. It allows for massive scalability across hundreds or thousands of servers in a *Hadoop* cluster.

The MapReduce algorithm consists of two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key-value-pairs). Reduce takes the output from a map as an input and combines those data tuples into smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

During a MapReduce job, *Hadoop* assigns the Map and Reduce tasks to the appropriate serves in the cluster. The framework manages all the details of data passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes.

Most of the computing takes place on nodes with data on local disks that reduces the network traffic.

After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the *Hadoop* server.

2.12 Spark Cluster

Spark [35] is an open source processing engine that provides scalable, massively parallel, in-memory execution environment for running analytics applications. It can be thought as an in-memory layer that sits above multiple data stores, where data can be loaded into memory and analyzed in parallel across a cluster.

Much like MapReduce, Spark works to distribute data across a cluster, and process that data in parallel. But unlike MapReduce - which shuffles files around on disk - Spark works in memory, making it much faster at processing data than MapReduce.

Spark includes rebuilt machine-learning algorithms and graph analysis algorithms that are especially written to execute in parallel and in memory. It also supports interactive SQL processing of queries and real-time streaming analytics.

Spark provides programmers with an application programming interface (API) centered on a data structure called the resilient distributed dataset (RDD), a read-only multiset of data items distributed over a cluster of machines, that is maintained in a fault-tolerant way. The availability of RDDs facilitates the implementation of both iterative algorithms, that visit their dataset multiple times in a loop, and exploratory data analysis (the repeated database-style querying of data). Thus, Spark is the ideal platform for implementing pragmatic PPCA (our proposed algorithm) which is an iterative algorithm.

2.13 Dimensionality Reduction

2.14 Vector Norms

A norm is a function $||.|| : V \rightarrow R$ which satisfies

(I) $\parallel x \parallel \geq 0, \forall x \in V \; and \parallel x \parallel = 0 \iff x = 0$

(II) $\parallel x + y \parallel \leq \parallel x \parallel + \parallel y \parallel, \forall x, y \in V$

(III) $\parallel \lambda x \parallel = \parallel \lambda \parallel \parallel x \parallel, \forall \lambda \in C \quad and \quad \forall x \in V$

Where $V \subseteq C^n$ is known as the vector space. In words, these conditions require that (I) the norm of a nonzero vector is strictly positive, (II) the norm of a vector sum does not exceed the sum of the norms of its parts which is known as the triangle inequality, and (III) scaling a vector scales its norm by the same amount. Thus, norms can be thought of as functions that define the size of a vector.

2.15 p-Norm

p-Norm is a family of vector norms, denoted as $\| \, . \, \|_p$, given by:

$$\| \, x \, \|_p \quad = \quad (\sum_{i=1}^{n} |x_i|^p)^{\frac{1}{p}}$$

The most widely used are the 1-norm, 2-norm (also known as Euclidean norm), and $\infty - norm$ (also known as sup-norm):

$$\| \, x \, \|_1 = \sum_{i=1}^{n} |x_i|$$

$$\| \, x \, \|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$$

$$\| \, x \, \|_\infty = max(|x_i|)$$

2.16 Frobenius Norm

This is a element-wise norm where the vector norm used is the 2-norm. Each entry of the matrix is considered as a dimension of a vector and 2-norm of the resulting vector is calculated. So we have,

$$\| \, A \, \|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^2}$$

which is equivalent to

$$\| \, A \, \|_F = \sqrt{tr(A * A)} = \sqrt{tr(AA*)}$$

where A is an mxn matrix, a_ij is the value in the i-th row and j-th column of A, A* is the conjugate transpose of A, and tr(B) is the trace of B (the sum of its diagonal entries). Since tr(AA*) is the sum of the eigenvalues of AA*, we have an alternative characterization of the Frobenius norm:

$$\| \, A \, \|_F = \sqrt{\sum_{i=1}^{n} \lambda_i}$$

2.17 Normal Distribution

In probability theory, the normal (or Gaussian) distribution is a very common continuous probability distribution. Normal distributions are important in statistics and are often used in the natural and social sciences to represent real-valued random variables whose distributions are not known [2] [2].

The normal distribution is useful because of the central limit theorem. In its most general form, under some conditions (which include finite variance), it states that averages of samples of observations of random variables independently drawn from independent distributions converge in distribution to the normal, that is, become normally distributed when the number of observations is sufficiently large. Physical quantities that are expected to be the sum of many independent processes (such as measurement errors) often have distributions that are nearly normal [3] Moreover, many results and methods (such as propagation of uncertainty and least squares parameter fitting) can be derived analytically in explicit form when the relevant variables are normally distributed.

The normal distribution is sometimes informally called the bell curve. However, many other distributions are bell-shaped (such as the Cauchy, Student's t, and logistic distributions). Even the term Gaussian bell curve is ambiguous because it may be used to refer to some function defined in terms of the Gaussian function which is not a probability distribution because it is not normalized in that it does not integrate to 1.

The probability density of the normal distribution is [5]

$$f(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Where:

- μ is the mean or expectation of the distribution (and also its median and mode).
- σ is the standard deviation.
- σ^2 is the variance.

2.18 Matrix Diagonalization

Matrix diagonalization is the process of taking a square matrix and converting it into a special type of matrix–a so-called diagonal matrix–that shares the same fundamental properties of the underlying matrix. Matrix diagonalization is equivalent to transforming the underlying system

of equations into a special set of coordinate axes in which the matrix takes this canonical form. Diagonalizing a matrix is also equivalent to finding the matrix's eigenvalues, which turn out to be precisely the entries of the diagonalized matrix. Similarly, the eigenvectors make up the new set of axes corresponding to the diagonal matrix.

The remarkable relationship between a diagonalized matrix, eigenvalues, and eigenvectors follows from the beautiful mathematical identity (the eigen decomposition) that a square matrix A can be decomposed into the very special form

$$A \quad = \quad PDP^{-1}$$

where P is a matrix composed of the eigenvectors of A, D is the diagonal matrix constructed from the corresponding eigenvalues, and P^{-1} is the matrix inverse of P. According to the eigen decomposition theorem, an initial matrix equation

$$AX \quad = \quad Y$$

can always be written

$$PDP^{-1}X = Y$$

(at least as long as P is a square matrix), and premultiplying both sides by P^{-1} gives

$DP^{-}1X = P^{-1}Y.$

Since the same linear transformation P^{-1} is being applied to both X and Y, solving the original system is equivalent to solving the transformed system

$DX' = Y'$,

where $X' = P^{-1}X$ and $Y' = P^{-1}Y$. This provides a way to canonicalize a system into the simplest possible form, reduce the number of parameters from $n \times n$ for an arbitrary matrix to n for a diagonal matrix, and obtain the characteristic properties of the initial matrix. This approach arises frequently in physics and engineering, where the technique is oft used and extremely powerful.

2.19 Expectation Maximization (EM) Algorithm

2.20 Stop Condition

In iterative algorithm like EM, main terminating condition considered is the change in desired variable or objective function. For example in PPCA our desired variables are W and σ^2. To check convergence we can give tolerance limit as input. If magnitude of changes in desired variable is less than our tolerance limit then we consider our algorithm has converged. But this is not most effective when we want to know how much accuracy we have obtained after convergence. Thus in many cases, error is checked after each iteration and algorithm is terminated if error goes down our target limit.

Chapter 3

PCA as a Data Analytic Tool

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components [20]. The number of principal components is less than or equal to the smaller of the number of original variables or the number of observations. This transformation is defined in such a way that the first principal component has the largest possible variance and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. As we can use PCA for dimensionality reduction, it can be used as a preprocessing step in many machine learning algorithms that do not perform well with high-dimensional data. Therefore we can easily realize that PCA is an excellent tool for big data analysis.

There are several ways to perform PCA. Eigen Value Decomposition (EVD) of covariance matrix and Singular Value Decomposition (SVD) are two basic and most common ways [21]. These methods usually perform better on small datasets on a single machine. But in distributed settings and for big data, they introduce a new set of difficulties; they do not scale well to high dimensional data and are inefficient in terms of computation and communication cost [22]. To overcome these difficulties, two other methods, Stochastic SVD (SSVD) [23] and Probabilistic PCA (PPCA) [24] are used in practice. But unlike the other methods to calculate principal component (i.e. EVD of covariance matrix, SVD, SSVD), PPCA has two important advantages. It has the ability of handling missing data and calculating probabilistic model to generate synthesized data [25]. These features are best suited for big data as missing values are prevalent in this case and a probabilistic model of the data will be more effective for analytical purposes.

Despite of being an excellent tool of data analysis, we lack in proper approach of PPCA in big data analysis. We saw a decent approach of making scalable PCA or *sPCA* in a distributed environment [22]. It was designed for data analysis on large datasets on distributed commodity clusters. But unfortunately this approach is not capable of handling datasets with very large

dimension, we can say data that is at the same time tall and wide. Moreover data sovereignty is not ensured. There was no method of performing PCA on big data located at different geographic location which involves independent computation at different clusters and accumulation of intermediate results.

In this chapter, we will try to show in depth the various approaches of performing PCA with their advantages and disadvantages. Moreover we will show the very recent optimization on scalability of PPCA which is known as *sPCA* with its special features and limitations.

3.1 Assumptions Involved in PCA

Mathematically, PCA is defined as on orthogonal linear transformation that transforms the data to a new coordinate system such that the greatest variance by some projection of the data comes to lie on the first coordinate (called the first principal component), the second gratest variance on the second coordinate and so on. Therefore, the assumptions involved in PCA are:

1. **Linearity**
   Linearity vastly simplifies the problem by restricting the set of potential bases. With this assumption PCA is now limited to re-expressing the data as a linear combination of its basis vectors that is, we need to find the appropriate change of basis.
2. **Large variances have important structure**
   This assumption also encompasses the belief that the data has a high Signal-to-Noise Ratio ($SNR = \frac{\sigma^2_{signal}}{\sigma^2_{noise}}$). Hence, principal components with larger associated variances represent interesting structure, while those with lower variances represent noise. Note that this is a strong, and sometimes, incorrect assumption.
3. **Orthogonal PCs**
   This assumption provides an intuitive simplification that makes PCA soluble with linear algebra decomposition techniques.

3.2 PCA and Change of Basis

The goal of principal component analysis is to identify the most meaningful basis to re-express a data set. The hope is that this new basis will filter out the noise and reveal hidden structure. Determining this fact allows an experimenter to discern which dynamics are important, redundant or noise.

With this rigor we may now state more precisely what PCA asks: *Is there another basis, which is a linear combination of the original basis, that best re-expresses our data set?*

Let $\boldsymbol{X}$ be the original data set, where each column is a single sample (or moment in time) of our data set.Here $\boldsymbol{X}$ is a $D \times N$ matrix where $\boldsymbol{N}$ in the number of samples and $\boldsymbol{D}$ is the dimension of each sample. Let Y be another $D \times N$ matrix related by a linear transformation $\boldsymbol{P}$. $\boldsymbol{X}$ is the original recorded data set and $\boldsymbol{Y}$ is a new representation of that data set.

$$\boldsymbol{Y} = \boldsymbol{PX} \tag{3.1}$$

Also let us define the following quantities

- $\boldsymbol{p_i}$ are the rows of $\boldsymbol{P}$
- $\boldsymbol{x_i}$ are the columns of $\boldsymbol{X}$
- $\boldsymbol{y_i}$ are the columns of $\boldsymbol{Y}$

Equation 3.1 represents a change of basis and thus can have many interpretations:

- $\boldsymbol{P}$ is a matrix that transforms $\boldsymbol{X}$ into $\boldsymbol{Y}$
- Geometrically, $\boldsymbol{P}$ is a rotation and a stretch which again transforms $\boldsymbol{X}$ into $\boldsymbol{Y}$
- The rows of $\boldsymbol{P}$, $\{\boldsymbol{p_1}, \boldsymbol{p_2}, \ldots, \boldsymbol{p_D}\}$ are a set of new basis vectors for expressing the columns of $\boldsymbol{X}$

The latter interpretation is not obvious but can be seen by writing out the explicit dot products of $\boldsymbol{PX}$

$$\boldsymbol{PX} = \begin{bmatrix} \boldsymbol{p_1} \\ \vdots \\ \boldsymbol{p_D} \end{bmatrix} \begin{bmatrix} \boldsymbol{x_1} & \ldots & \boldsymbol{x_N} \end{bmatrix}$$

$$\boldsymbol{Y} = \begin{bmatrix} \boldsymbol{p_1} \cdot \boldsymbol{x_1} & \ldots & \boldsymbol{p_1} \cdot \boldsymbol{x_N} \\ \vdots & \ddots & \vdots \\ \boldsymbol{p_D} \cdot \boldsymbol{x_1} & \ldots & \boldsymbol{p_N} \cdot \boldsymbol{x_N} \end{bmatrix}$$

We can note the form of each column of $\boldsymbol{Y}$

$$\boldsymbol{y_i} = \begin{bmatrix} \boldsymbol{p_1} \cdot \boldsymbol{x_i} \\ \vdots \\ \boldsymbol{p_D} \cdot \boldsymbol{x_i} \end{bmatrix}$$

We recognize that each coefficient of $\boldsymbol{y_i}$ is a dot-product of $\boldsymbol{x_i}$ with the corresponding row in $\boldsymbol{P}$. In other words, the j^{th} coefficient of $\boldsymbol{y_i}$ is a projection on to the j^{th} row of $\boldsymbol{P}$. This is in fact the very form of an equation where $\boldsymbol{y_i}$ is a projection on to the basis of $\{\boldsymbol{p_1}, \ldots, \boldsymbol{p_D}\}$. Therefore, the rows of P are a new set of basis vectors for representing of columns of $\boldsymbol{X}$.

By assuming linearity the problem reduces to finding the appropriate change of basis. The row vectors $\{\boldsymbol{p_1}, \ldots, \boldsymbol{p_D}\}$ in this transformation will become the principal components of $\boldsymbol{X}$.. Several questions now arise:

- What is the best way to re-express $\boldsymbol{X}$?
- What is a good choice of basis $\boldsymbol{P}$?

These questions must be answered by next asking ourselves what features we would like $\boldsymbol{Y}$ to exhibit. Evidently, additional assumptions beyond linearity are required to arrive at a reasonable result. The selection of these assumptions is the subject of the next section.

3.3 Eigenvalue Decomposition (EVD)

Given a matrix $\mathbf{Y}$ of size $N \times D$ (N rows and D columns), a PCA algorithm obtains d principal components ($d \leq D$) that explain the most variance (and hence information) of the data in matrix $\mathbf{Y}$ [26, 27]. To be useful in practice, d is chosen to be much smaller than D, that is $d \ll D$. In case EVD technique, the target matrix V is of dimension $N \times k$ where the columns of V are the principal components of $\mathbf{Y}$.

3.3.1 Covariance Matrix

How do we quantify and generalize these notions to arbitrarily higher dimensions? Consider two sets of measurements with zero means:

$$A = \{a_1, a_2, ..., a_N\}, B = \{b_1, b_2, ..., b_N\}$$

where the subscript denotes the sample number. The variance of $\boldsymbol{A}$ and $\boldsymbol{B}$ are individually defined as,

$$\sigma_A^2 = \frac{1}{N}\sum_i a_i^2$$

$$\sigma_B^2 = \frac{1}{N}\sum_i b_i^2$$

The covariance between $\boldsymbol{A}$ and $\boldsymbol{B}$ is a straight-forward generalization.

$$covariance\ of\ \boldsymbol{A}\ and\ \boldsymbol{B} \equiv \sigma_{AB}^2 = \frac{1}{N}\sum_i a_i b_i$$

The covariance measures the degree of the linear relationship between two variables. A large positive value indicates positively correlated data. Likewise, a large negative value denotes negatively correlated data. The absolute magnitude of the covariance measures the degree of redundancy. Some additional facts about the covariance:

- σ_{AB} is zero if and only if $\boldsymbol{A}$ and $\boldsymbol{B}$ are uncorrelated (e.g. Figure 3.1, left panel)
- $\sigma_{AB}^2 = \sigma_A^2\ if\ A = B$

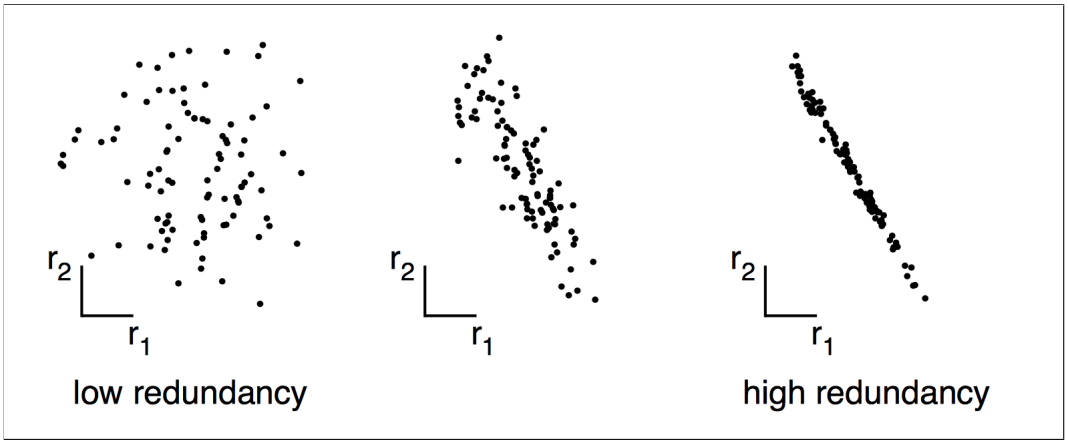


Figure 3.1: A spectrum of possible redundancies in data from the two separate measurements r_1 and r_2. The two measurements on the left are uncorrelated because one can not predict one from the other. Conversely, the two measurements on the right are highly correlated indicating highly redundant measurements.

We can equivalently convert $\boldsymbol{A}$ and $\boldsymbol{B}$ into corresponding row vectors.

$$\boldsymbol{a} = [a_1\ a_2\ \ldots\ a_n]$$
$$\boldsymbol{b} = [b_1\ b_2\ \ldots\ b_n]$$

so that we may express the covariance as a dot product matrix computation

$$\sigma_{ab}^2 \equiv \frac{1}{N}\boldsymbol{a}\boldsymbol{b}^T$$

Finally, we can generalize from two vectors to an arbitrary number. Rename the row vectors a and $\boldsymbol{b}$ as $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$, respectively, and consider additional indexed row vectors $\boldsymbol{x_1}$, ... , $\boldsymbol{x_D}$. Define a new $D \times N$ matrix $\boldsymbol{X}$.

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x_1} \\ \vdots \\ \boldsymbol{x_D} \end{bmatrix}$$

One interpretation of $\boldsymbol{X}$ is the following. Each row of $\boldsymbol{X}$ corresponds to all measurements of a particular type. Each column of $\boldsymbol{X}$ corresponds to a set of measurements from one particular trial. We now arrive at a definition for the covariance matrix $\boldsymbol{C_x}$

$$\boldsymbol{C_x} \equiv \frac{1}{\boldsymbol{N}} \boldsymbol{X} \boldsymbol{X}^T$$

Consider the matrix $\boldsymbol{C_x} = \frac{1}{\boldsymbol{N}} \boldsymbol{X} \boldsymbol{X}^T$ The ij^{th} element of $\boldsymbol{C_x}$ is the dot product between the vector of the ith measurement type with the vector of the j^{th} measurement type. We can summarize several properties of $\boldsymbol{C_x}$:

- $\boldsymbol{C_x}$ is a square symmetric $D \times D$ matrix (Theorem 2 of Appendix A)
- The diagonal terms of $\boldsymbol{C_x}$ are the variance of particular measurement types
- The off-diagonal terms of $\boldsymbol{C_x}$ are the covariance between measurement types.

$\boldsymbol{C_x}$ captures the covariance between all possible pairs of measurements. The covariance values reflect the noise and redundancy in our measurements.

- In the diagonal terms, by assumption, large values correspond to interesting structure
- In the off-diagonal terms large magnitudes correspond to high redundancy

Pretend we have the option of manipulating $\boldsymbol{C_x}$. We will suggestively define our manipulated covariance matrix $\boldsymbol{C_Y}$. What features do we want to optimize in $\boldsymbol{C_Y}$?

3.3.2 Diagonalize the Covariance Matrix

We can summarize the last two sections by stating that our goals are

1. to minimize redundancy, measured by the magnitude of the covariance
2. maximize the signal, measured by the variance

What would the optimized covariance matrix $\boldsymbol{C_Y}$ look like?

- All off-diagonal terms in $\boldsymbol{C_Y}$ should be zero. Thus, $\boldsymbol{C_Y}$ must be a diagonal matrix. Or, said another way, $\boldsymbol{Y}$ is decorrelated
- Each successive dimension in Y should be rank-ordered according to variance

There are many methods for diagonalizing $\boldsymbol{C_Y}$. It is curious to note that PCA arguably selects the easiest method: PCA assumes that all basis vectors $\{\boldsymbol{p_1}, \ldots, \boldsymbol{p_m}\}$ are orthonormal, i.e. $\boldsymbol{P}$ is an orthonormal matrix.

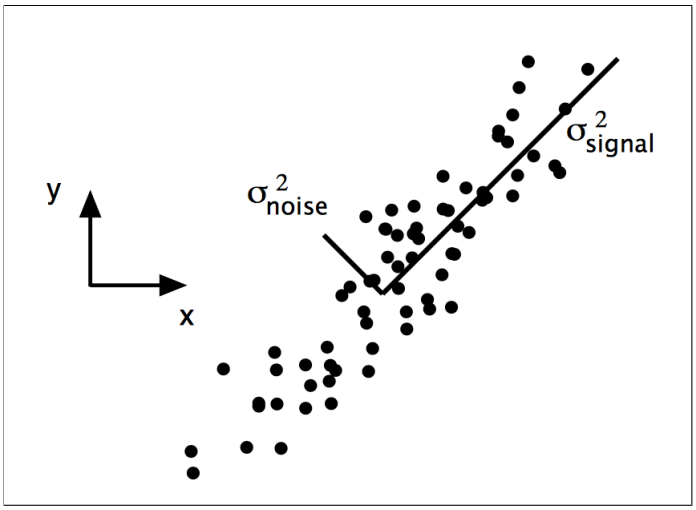


Figure 3.2: A typical 2D example. The signal and noise variances σ^2_{signal} and σ^2_{noise} are graphically represented by the two lines subtending the cloud of data. The largest direction of variance lie along the best-fit line

In a simple $2D$ example like in Figure 3.2, $\boldsymbol{P}$ acts as a generalized rotation to align a basis with the axis of maximal variance. In multiple dimensions this could be performed by a simple algorithm:

1. Select a normalized direction in m-dimensional space along which the variance in $\boldsymbol{X}$ is maximized. Save this vector as $\boldsymbol{p_1}$.
2. Find another direction along which variance is maximized, however, because of the orthonormality condition, restrict the search to all directions orthogonal to all previous selected directions. Save this vector as $\boldsymbol{p_2}$
3. Repeat this procedure until D vectors are selected.

The resulting ordered set of $\boldsymbol{p}$'s are the principal components.

3.3.3 Solving PCA in EVD Approach

We derive our first algebraic solution to PCA based on an important property of eigenvector decomposition. Once again, the data set is $\boldsymbol{P}$, an $D \times N$ matrix, where D is the number of

measurement types and N is the number of samples. The goal is summarized as follows

> Find some orthonormal matrix $\boldsymbol{P}$ in $\boldsymbol{Y} = \boldsymbol{PX}$ such that $\boldsymbol{C_Y} \equiv \frac{\mathbf{1}}{\boldsymbol{N}}\boldsymbol{XX}^T$ is a diagonal matrix. The rows of $\boldsymbol{P}$ are the principal components of $\boldsymbol{X}$

We begin by rewriting $\boldsymbol{C_Y}$ in terms of the unknown variable

$$\begin{aligned}
\boldsymbol{C_Y} &= \frac{1}{N}\boldsymbol{YY}^T \\
&= \frac{1}{N}(\boldsymbol{PX})(\boldsymbol{PX})^T \\
&= \frac{1}{N}\boldsymbol{PXX}^T\boldsymbol{P}^T \\
&= \boldsymbol{P}(\frac{1}{N}\boldsymbol{XX}^T)\boldsymbol{P}^T \\
\boldsymbol{C_Y} &= \boldsymbol{PC_X}\boldsymbol{P}^T
\end{aligned}$$

Note that we have identified the covariance matrix of $\boldsymbol{X}$ in the last line.

Our plan is to recognize that any symmetric matrix A is diagonalized by an orthogonal matrix of its eigenvectors (by Theorems 3 and 4 from Appendix A). For a symmetric matrix $\boldsymbol{A}$ Theorem 4 provides $\boldsymbol{A} = \boldsymbol{EDE}^T$, where $\boldsymbol{D}$ is a diagonal matrix and $\boldsymbol{E}$ is a matrix of eigenvectors of $\boldsymbol{A}$ arranged as columns.

Now we comes the trick. We select the matrix $\boldsymbol{P}$ to be a matrix where each row $\boldsymbol{p_i}$ is an eigenvector of $\frac{1}{N}\boldsymbol{XX}^T$. By this selection, $\boldsymbol{P} \equiv \boldsymbol{E}^T$. With this relation and Theorem 1 of Appendix A ($\boldsymbol{P}^{-1} = \boldsymbol{P}^T$) we can finish evaluating $\boldsymbol{C_Y}$.

$$\begin{aligned}
\boldsymbol{C_Y} &= \boldsymbol{PC_X}\boldsymbol{P}^T \\
&= \boldsymbol{P}(\boldsymbol{E}^T\boldsymbol{DE})\boldsymbol{P}^T \\
&= \boldsymbol{P}(\boldsymbol{P}^T\boldsymbol{DP})\boldsymbol{P}^T \\
&= (\boldsymbol{PP}^T)\boldsymbol{D}(\boldsymbol{PP})^T \\
&= (\boldsymbol{PP}^{-1})\boldsymbol{D}(\boldsymbol{PP})^{-1} \\
\boldsymbol{C_Y} &= \boldsymbol{D}
\end{aligned}$$

It is evident that the choice of $\boldsymbol{P}$ diagonalizes $\boldsymbol{C_Y}$. This was the goal for PCA. We can summarize the results of PCA in the matrices $\boldsymbol{P}$ and $\boldsymbol{C_Y}$.

- The principal components of $\boldsymbol{C_Y}$ are the eigenvectors of $\boldsymbol{C_X} = \frac{1}{N}\boldsymbol{XX}^T$

- The i^{th} diagonal value of $\boldsymbol{C_Y}$ is the variance of $\boldsymbol{X}$ along $\boldsymbol{p_i}$

3.4 Practical Approach of EVD

In practice computing PCA of a data set $\boldsymbol{X}$ is done in two steps:

1. subtracting off the mean of each measurement type
2. computing the eigenvectors of $\boldsymbol{C_X}$

3.5 Singular Value Decomposition (SVD)

Let **X** be an arbitrary $N \times D$ matrix and $\boldsymbol{X}^T\boldsymbol{X}$ be a rank r, square, symmetric $D \times D$ matrix.

3.5.1 Performing SVD

To do Singular Value Decomposition of matrix **X** let us assume that:

- $\{\hat{v}_1, \hat{v}_2, \ldots, \hat{v}_r\}$ is the set of orthonormal $D \times 1$ eigenvectors with associated eigenvalues $\{\lambda_1, \lambda_2, \ldots, \lambda_r\}$ for the symmetric matrix $\boldsymbol{X^TX}$

$$(\boldsymbol{X^TX})\hat{v}_i = \lambda_i\hat{v}_i$$

- $\sigma_i \equiv \sqrt{\lambda_i}$ are positive real and termed the singular values.
- $\{\hat{u}_1, \hat{u}_2, \ldots, \hat{u}_r\}$ is the set of $N \times 1$ vectors defined by $\boldsymbol{\hat{u}_i \equiv \frac{1}{\sigma_i} X\hat{v}_i}$

$$\boldsymbol{\sigma_i\hat{u}_i \equiv X\hat{v}_i} \tag{3.2}$$

This result says a quite a bit. $\boldsymbol{X}$ multiplied by an eigenvector of $\boldsymbol{X^TX}$ is equal to a scalar times another vector. We can summarize this result for all vectors in one matrix multiplication by following the prescribed construction in Figure 3.3. We start by constructing a new diagonal matrix $\sum$.

$$\sum \equiv \begin{bmatrix} \sigma_{\tilde{1}} & & & & \\ & \ddots & & \mathbf{0} & \\ & & \sigma_{\tilde{r}} & & \\ & & & 0 & \\ & \mathbf{0} & & \ddots & \\ & & & & 0 \end{bmatrix}$$

where $\sigma_{\tilde{1}} \geq \sigma_{\tilde{2}} \geq \cdots \geq \sigma_{\tilde{r}}$ are the rank-ordered set of singular values. Likewise we construct accompanying orthogonal matrices,

$$V = \begin{bmatrix} \hat{v}_{\tilde{1}} & \hat{v}_{\tilde{2}} & \dots & \hat{v}_{\tilde{D}} \end{bmatrix}$$

$$U = \begin{bmatrix} \hat{u}_{\tilde{1}} & \hat{u}_{\tilde{2}} & \dots & \hat{u}_{\tilde{N}} \end{bmatrix}$$

where we have appended an additional $(D - r)$ and $(N-$r) orthonormal vectors to "fill up" the matrices for **V** and **U** respectively (i.e. to deal with degeneracy issues). Figure 3.3 provides a graphical representation of how all of the pieces fit together to form the matrix version of *SVD*.

$$XV = U\sum$$

where each column of V and U perform the scalar version of the decomposition (Equation 3.2). Because V is orthogonal, we can multiply both sides by $V^{\mathbf{1}} = V^T$ to arrive at the final form of the decomposition.

$$X = U\sum V^T \tag{3.3}$$

Although derived without motivation, this decomposition is quite powerful. Equation 3.3 states that any arbitrary matrix **X** can be converted to an orthogonal matrix, a diagonal matrix and another orthogonal matrix (or a rotation, a stretch and a second rotation)

3.5.2 Linking SVD with EVD

How does this link in to the previous *EVD* analysis of PCA? Let us consider the $N \times D$ matrix, **X**, for which we have a singular value decomposition, $X = U\sum V^T$. There is a theorem from linear algebra which says that the non-zero singular values of **X** are the square roots of the nonzero eigenvalues of AA^T or A^TA.

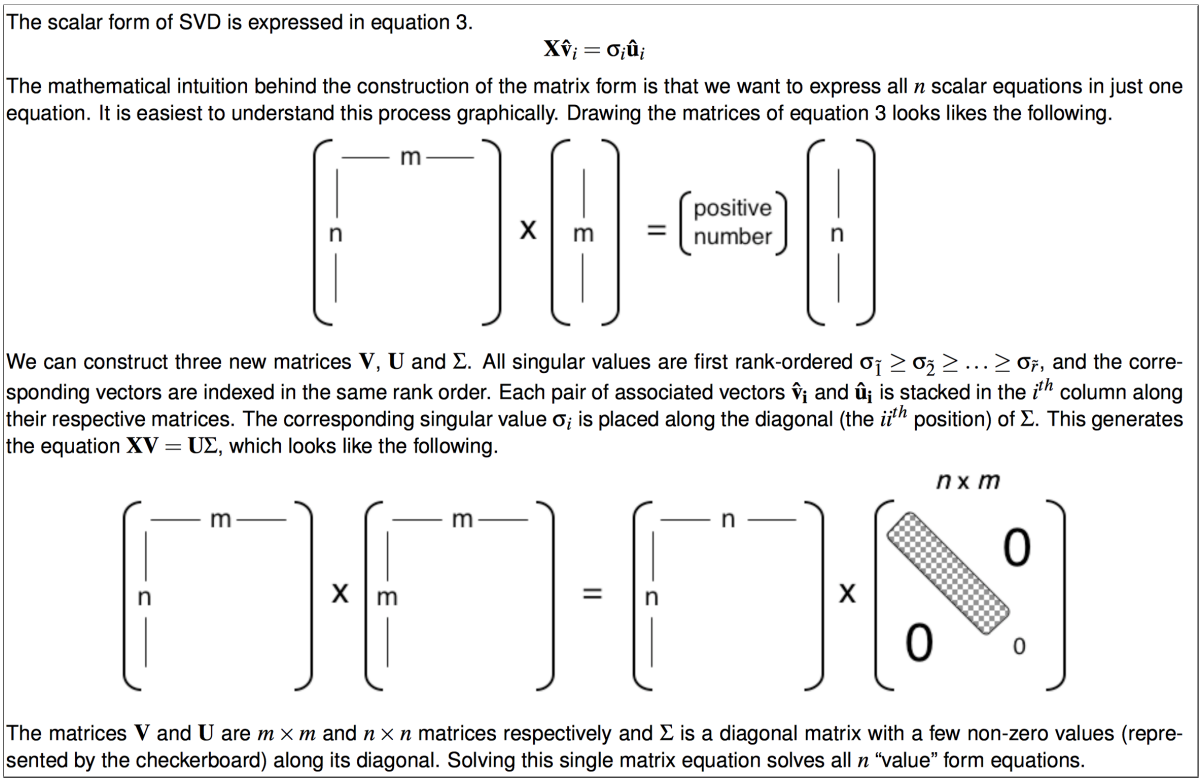


Figure 3.3: Construction of the matrix form of SVD 3.3 from the scalar form 3.2

The former assertion for the case $\boldsymbol{A}^T\boldsymbol{A}$ is proven in the following way:

$$\begin{aligned}\boldsymbol{A}^T\boldsymbol{A} &= (\boldsymbol{U}\sum\boldsymbol{V}^T)^T(\boldsymbol{U}\sum\boldsymbol{V}^T) \\ &= (\boldsymbol{V}\sum{}^T\boldsymbol{U}^T)(\boldsymbol{U}\sum\boldsymbol{V}^T) \\ &= \boldsymbol{V}(\sum{}^T\sum)\boldsymbol{V}^T\end{aligned}$$

We observe that $\boldsymbol{A}^T\boldsymbol{A}$ is similar to $\sum^T\sum$ and thus it has the same eigenvalues. Since $\sum^T\sum$ is a square $(D \times D)$, diagonal matrix, the eigenvalues are in fact the diagonal entries, which are the squares of the singular values. Note that the non-zero eigenvalues of each of the covariance matrices, $\boldsymbol{A}\boldsymbol{A}^T$ and $\boldsymbol{A}^T\boldsymbol{A}$ are actually identical.

It should also be noted that we have effectively performed an eigenvalue decomposition for the matrix, $\boldsymbol{A}^T\boldsymbol{A}$. Indeed, since AT A is symmetric, this is an orthogonal diagonalisation and thus the eigenvectors of $\boldsymbol{A}^T\boldsymbol{A}$ are the columns of $\boldsymbol{V}$. This will be important in making the practical connection between the SVD and and the PCA of matrix X, which is what we will do next.

Returning to the original $N \times D$ data matrix, $\boldsymbol{X}$, let us define a new $D \times N$ matrix, $\boldsymbol{Z}$:

$$\boldsymbol{Z} = \frac{1}{\sqrt{N-1}}\boldsymbol{X}^T$$

Recall that since the m rows of $\boldsymbol{X}$ contained the N data samples, we subtracted the row average from each entry to ensure zero mean across the rows. Thus, the new matrix, $\boldsymbol{Z}$ has columns with zero mean. Consider forming the $D \times D$ matrix, $\boldsymbol{Z}^T\boldsymbol{Z}$:

$$\begin{aligned}\boldsymbol{Z}^T\boldsymbol{Z} &= (\frac{1}{\sqrt{N-1}}\boldsymbol{X}^T)^T(\frac{1}{\sqrt{N-1}}\boldsymbol{X}^T) \\ &= \frac{1}{N-1}\boldsymbol{X}\boldsymbol{X}^T \\ i.e.\ \boldsymbol{Z}^T\boldsymbol{Z} &= \boldsymbol{C_x}\end{aligned}$$

We find that defining $\boldsymbol{Z}$ in this way ensures that $\boldsymbol{Z}^T\boldsymbol{Z}$ is equal to the covariance matrix of $\boldsymbol{Z}$, $\boldsymbol{C_x}$. From the discussion in the previous section, the principal components of $\boldsymbol{X}$ (which is what we are trying to identify) are the eigenvectors of $\boldsymbol{C_x}$. Therefore, if we perform a singular value decomposition of the matrix $\boldsymbol{Z}^T\boldsymbol{Z}$, the principal components will be the columns of the orthogonal matrix, $\boldsymbol{V}$.

The last step is to relate the *SVD* of $\boldsymbol{Z}^T\boldsymbol{Z}$ back to the change of basis:

$$\boldsymbol{Y} = \boldsymbol{P}\boldsymbol{X}$$

We wish to project the original data onto the directions described by the principal components. Since we have the relation $\boldsymbol{V} = \boldsymbol{P}^T$, this is simply:

$$\boldsymbol{Y} = \boldsymbol{V}^T\boldsymbol{X}$$

If we wish to recover the original data, we simply compute (using orthogonality of $\boldsymbol{V}$):

$$\boldsymbol{X} = \boldsymbol{V}\boldsymbol{Y}$$

3.5.3 SVD for Dense Matrices

Golub and Kahan [28] introduced a two-step approach for computing SVD: convert the input matrix to a bidiagonal one and then perform SVD on the bidiagonal matrix. Demmel and Kahan [29] improved this approach by adding another step before bidiagonalization, which is QR decomposition. [22] refered to this method as SVD-Bidiag, which has the following three steps for a given matrix $\boldsymbol{Y}$:

1. compute the QR decomposition of $\boldsymbol{Y}$, which results in an orthogonal matrix $\mathbf{Q}$ and an upper triangular matrix $\boldsymbol{R}$

2. transform **R** to a bidiagonal matrix B
3. compute SVD on B

The SVD-Bidiag algorithm is implemented in *RScaLAPACK*. Their analysis showed that the computational complexity of the SVD-Bidiag algorithm is dominated by the QR decomposition and bi-diagonalization steps, and is given by $\mathcal{O}(ND^2 + D^3)$. Therefore, the SVD-Bidiag algorithm is only suitable when D is small. More details can be found in [22].

3.5.4 SVD for Sparse Matrices

SVD can be computed efficiently for sparse matrices using Lanczos' algorithm [30], which has a computational complexity of $\mathcal{O}(Nz^2)$, where z is the number of non-zero dimensions (out of **D** dimensions). More details can be found in [22].

3.6 Stochastic SVD (SSVD)

Randomized sampling techniques have recently gained popularity in solving large-scale linear algebra problems. The work in [31] describes a randomized method to compute approximate decomposition of matrices, which is referred to as stochastic SVD (SSVD). SSVD has two steps:

1. It uses randomized techniques to compute a low-dimensional approximation of the input matrix
2. It performs SVD on the approximation matrix

The accuracy of the results depends on the performance of the randomized techniques and the size of the approximation matrix. Accuracy can be improved through running the randomization step multiple times. Therefore, SSVD has the flexibility of trading off the accuracy of the results with the required computational resources.

3.7 Computational Complexity

Computational complexity of SSVD is dominated by the first step, which is $\mathcal{O}(DNd)$. This is a much better complexity than the previous techniques, because d is typically much smaller than D and is usually a constant. However, SSVD requires exchanging multiple intermediate matrices, which may cause a problem for scalability. The amount of intermediate data can be up to $\mathcal{O}(max(Nd, d2))$ [22].

3.8 Probabilistic PCA (PPCA)

We will be trying to present *PPCA* in some kind of detail. For this we will follow the research work of Tipping and Bishop [32]

3.8.1 What is PPCA

We can obtain a probabilistic formulation of PCA by introducing a Gaussian latent i.e. unobserved variable model. This kind of model is highly related to statistical factor analysis. First let us define some quantity:

- $\boldsymbol{y}$ is a D dimensional observed data vector
- $\boldsymbol{x}$ is a d dimensional latent variable
- $\boldsymbol{W}$ is a $D \times d$ transformation matrix
- the columns of $\boldsymbol{W}$ are the principal components
- $\boldsymbol{\mu}$ is the dimension wise mean vector of $\boldsymbol{y}$. This parameter allows the data to have non zero mean
- $\boldsymbol{\epsilon}$ is a D-dimensional zero-mean noise variable
- Here $\boldsymbol{x}, \boldsymbol{\epsilon}, \boldsymbol{y}$ are normal distributed i.e. Gaussian distributed

$$\begin{aligned} \boldsymbol{x} &\sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}) \\ \boldsymbol{\epsilon} &\sim \mathcal{N}(\mathbf{0}, \boldsymbol{\sigma^2 I}) \\ \boldsymbol{y} &\sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{W W}^T + \boldsymbol{\sigma^2}) \end{aligned}$$

  where $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{u}, \sum)$ denotes the Normal distribution with $\boldsymbol{u}$ mean and $\sum$ covariance matrix.

A latent variable model seeks to relate a D-dimensional observation vector $\boldsymbol{y}$ to a corresponding d-dimensional vector of latent variables $\boldsymbol{x}$ where the relationship is linear:

$$\boldsymbol{y} = \boldsymbol{W x} + \boldsymbol{\mu} + \boldsymbol{\epsilon} \tag{3.4}$$

The motivation is that, with $d < D$, the latent variables will offer a more parsimonious explanation of the dependencies between the observations. The model parameters may thus be determined by maximum-likelihood, As there is no closed-form analytic solution for finding $\boldsymbol{W}$ and $\boldsymbol{\sigma}^2$, their values must be obtained via an iterative procedure.

3.8.2 The Probability Model

The use of the isotropic Gaussian noise model $\mathcal{N}(\mathbf{0}, \boldsymbol{\sigma^2 I})$ for in conjunction with equation (3.4) implies that the $\boldsymbol{x}$-conditional probability distribution over $\boldsymbol{y}$-space is given by:

$$\boldsymbol{y}|\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{\mu}, \boldsymbol{\sigma}^2\boldsymbol{I}) \tag{3.5}$$

With the marginal distribution over the latent variables also Gaussian and conventionally defined by $\boldsymbol{x} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, the marginal distribution for the observed data $\boldsymbol{y}$ is readily obtained by integrating out the latent variables and is likewise Gaussian:

$$\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{C}) \tag{3.6}$$

where the observation covariance model is specified by $\boldsymbol{C} = \boldsymbol{W}\boldsymbol{W}^T + \boldsymbol{\sigma^2}$. given N observations $\{\boldsymbol{y_n}\}_1^N$ as the input data, the log likelihood of data is given by:

$$\begin{aligned} \mathcal{L}(\{\boldsymbol{y_r}\}_1^N) &= \sum_{n=1}^{N} \ln p(\boldsymbol{n_r}) \\ &= -\frac{1}{N}\{D * \ln(2\pi) + \ln|\boldsymbol{C}| + tr(\boldsymbol{C}^{-1} * \boldsymbol{S})\} \end{aligned} \tag{3.7}$$

where $\boldsymbol{S}$ is the sample covariance matrix of data $\{\boldsymbol{y_r}\}$ given by:

$$\boldsymbol{S} = \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{y_r} - \boldsymbol{\mu})(\boldsymbol{y_r} - \boldsymbol{\mu})^T \tag{3.8}$$

and $tr(\boldsymbol{M})$ is the trace of matrix $\boldsymbol{M}$

3.8.3 Properties of the Maximum-Likelihood Estimators

According to [32], the likelihood equation 3.7 is maximized when:

$$\boldsymbol{W}_{ML} = \boldsymbol{U}_d\sqrt{\boldsymbol{\Lambda}_d - \sigma^2\boldsymbol{I}}\boldsymbol{R} \tag{3.9}$$

where:

where the d column vectors in the $D \times d$ matrix $\boldsymbol{U}_d$ are the principal eigenvectors of $\boldsymbol{S}_d$, with corresponding eigenvalues $\{\lambda_1, \ldots, \lambda_d\}$ in the $d \times d$ diagonal matrix $\boldsymbol{\Lambda}_d$, and $\boldsymbol{R}$ is an arbitrary $d \times d$ orthogonal rotation matrix. Thus, from Equation (3.9), the latent variable model defined by Equation (3.4) effects a mapping from the latent space into the principal subspace of the observed data.

It may also be shown that for $\boldsymbol{W} = \boldsymbol{W}_{ML}$, the maximum-likelihood estimator for σ^2 is given by:

$$\sigma^2_{ML} = \frac{1}{D-d}\sum_{i=d+1}^{D}\lambda_i \tag{3.10}$$

which has a clear interpretation as the variance 'lost' in the projection, averaged over the lost dimensions.

3.8.4 An EM Algorithm for Probabilistic PCA

In the EM approach to maximising the likelihood for PPCA, we consider the latent variables $\{\boldsymbol{x}_n\}$ to be 'missing' data and the 'complete' data to comprise the observations together with these latent variables. The corresponding complete-data log-likelihood is then:

$$\mathcal{L}_C = \sum_{n=1}^{N}\ln p(\boldsymbol{y}_n), \boldsymbol{x}_n) \tag{3.11}$$

The corresponding E-step and M-step are given below:

$$\textbf{E-step:} \qquad \langle \boldsymbol{x}_n \rangle = \boldsymbol{M}^{-1}\boldsymbol{W}^T(\boldsymbol{y}_n - \boldsymbol{\mu}) \tag{3.12}$$

$$\textbf{E-step:} \qquad \langle \boldsymbol{x}_n \boldsymbol{x}_n^T \rangle = \sigma^2 \boldsymbol{M}^{-1} + \langle \boldsymbol{x}_n \rangle \langle \boldsymbol{x}_n \rangle^T \tag{3.13}$$

$$\textbf{M-step:} \qquad \tilde{\boldsymbol{W}} = \left[\sum_{n=1}^{N}(\boldsymbol{y}_n - \boldsymbol{\mu})\langle \boldsymbol{x}_n \rangle^T\right]\left[\sum_{n=1}^{N}\langle \boldsymbol{x}_n \boldsymbol{x}_n^T \rangle\right]^{-1} \tag{3.14}$$

$$\textbf{M-step:} \qquad \sigma^2 = \frac{1}{ND}\sum_{n=1}^{N}\left[||\boldsymbol{y}_n - \boldsymbol{\mu}||^2 - 2\langle \boldsymbol{x}_n \rangle \widetilde{\boldsymbol{W}}(\boldsymbol{y}_n - \boldsymbol{\mu}) + tr\left(\langle \boldsymbol{x}_n \boldsymbol{x}_n^T \rangle \widetilde{\boldsymbol{W}}^T \widetilde{\boldsymbol{W}}\right)\right] \tag{3.15}$$

In order to make the EM algorithm look simple we define:

$$\begin{aligned} \boldsymbol{E}_n &= \langle \boldsymbol{x}_n \rangle \\ \boldsymbol{F}_n &= \langle \boldsymbol{x}_n \boldsymbol{x}_n^T \rangle \\ \boldsymbol{Y}_m &= (\boldsymbol{y}_n - \boldsymbol{\mu}) \\ Frob(\boldsymbol{Y}_m) &= ||\boldsymbol{y}_n - \boldsymbol{\mu}||^2 \end{aligned}$$

Therefore, we can write the EM steps with simplified notations:

$$\textbf{E-step:} \quad \boldsymbol{E} = \boldsymbol{M}^{-1}\boldsymbol{W}^T(\boldsymbol{Y}_m)$$

$$\textbf{E-step:} \quad \boldsymbol{F} = \sigma^2\boldsymbol{M}^{-1} + \boldsymbol{E}\boldsymbol{E}^T$$

$$\textbf{M-step:} \quad \widetilde{\boldsymbol{W}} = (\boldsymbol{Y}_m\boldsymbol{E}^T)\boldsymbol{F}^{-1}$$

$$\textbf{M-step:} \quad \sigma^2 = \frac{1}{ND}\left[Frob(\boldsymbol{Y}_m) - 2\sum_{n=1}^{N}\boldsymbol{E}_n^T\widetilde{\boldsymbol{W}}\boldsymbol{Y}_{mn} + tr\left(\boldsymbol{F}\widetilde{\boldsymbol{W}}^T\widetilde{\boldsymbol{W}}\right)\right]$$

Iterating Equation (3.12), (3.13), (3.14), (3.15) upto convergence, we can make a good approximation of both $\boldsymbol{W}$ and $\boldsymbol{\sigma^2}$. The basic algorithm of PPCA is given below:

Algorithm 1 Basic PPCA

$W = normrnd(D, d)$
$ss = normrnd(1, 1)$
$Y_m = columnMean(Y)$
$Y_c = Y - Ym$
while $(!Stop\_Condition)$ **do**
  $M = W^T * W + ss * I$
  $X = Y_c * W * M^{-1}$
  $XtX = X^T * X + ss * M^{-1}$ §TATE $YtX = Y_c^T * X$
  $W = YtX \times$ *invert(*XtX*)*
  $ss2 = trace(XtX * W^T * W)$
  $ss3 = \sum_{n=1}^{N} X_n * W^T * Y_{cn}$
  $ss = (||Yc||_F^2 + ss2 - 2 * ss3) \times N^{-1} \times D^{-1}$
end while

3.9 Recent Implementation of *sPCA*

[22] presented a scalable implementation of Probabilistic PCA for distributed platforms such as MapReduce and Spark.

3.9.1 Special Features

The implementation of *sPCA* has a good number of features:

Mean Propagation to Leverage Sparsity

The first optimization they propose is the mean propagation idea, which preserves and utilizes the sparsity of the input matrix $\boldsymbol{Y}$. PPCA requires the input matrix to be mean-centered, meaning that the mean vector $\boldsymbol{\mu}$ must be subtracted from each row of the original matrix $\boldsymbol{Y}$. Large matrices, however, are mostly sparse, with many zero elements. Sparse matrices can achieve a small disk and memory footprint by storing only non-zero elements, and performing operations only over non-zero elements. Subtracting the non-zero mean from the matrix would make many elements non-zero, so the advantage of sparsity is lost.

To avoid the problems of subtracting the mean, they keep the original matrix $\boldsymbol{Y}$ and the mean $\boldsymbol{\mu}$ in two separate data structures. they did not subtract the mean $\boldsymbol{\mu}$ from $\boldsymbol{Y}$. Rather, they propagate the mean throughout the different matrix operations.

Minimizing Intermediate Data

Intermediate data can slow down the distributed execution of any PCA algorithm, because it needs to be transferred to other nodes for processing to continue. Their second optimization is job consolidation, which means merging multiple distributed jobs into one in order to reduce the communication between these jobs.

Efficient Matrix Multiplication

PPCA requires many matrix multiplications, which are expensive operations in a distributed setting. To appreciate the techniques that *sPCA* employs to overcome the inefficiency of matrix multiplication, they briefly explain different possible implementations of this operation.

Efficient Frobenius Norm Computation

The PPCA algorithm requires computing the Frobenius norm of the mean-centered input matrix. To solve this problem, they design an algorithm which does not even require creating the dense vector. Many machine learning algorithms compute various norms of matrices. The proposed method for optimizing the computation of the Frobenius norm can be extended to other matrix norms using similar ideas. Thus, this simple optimization can benefit several other machine learning algorithms.

3.9.2 Limitations

Despite of having some wonderful features *sPCA* has some limitations also. We can discuss these limitations from two aspects.

1. The approach is not applicable for high dimensional data or we can say tall and wide big data. Though the approach has done some efficient use of memory by reducing the generation of intermediate data, it still runs out of memory while running on big data whose vertical dimension is in the range of millions. We failed to run *sPCA* on dataset of dimension $10M \times 5M$ in our set-up cluster where we have machines with $64GB$ of RAM.

2. There was no implementation on geographically distributed big data. In current world where data is distributed across national border to ensure fast access and national data sovereignty, it is necessary to be capable of doing data analysis on such geo-distributed data. However, *sPCA* did not indicate a process of generating some kind of partial result and later accumulate them to produce the final analytical results.

Chapter 4

Our Focus

In current world, the rate of generation of data is quite large. We live in an age of Big Data and Internet of Things (loT). The term Big Data is not really an overstatement. Numerous web services share and collect data of huge scale, typically in terabytes range, and the data includes various aspects of users, for example, clicks, visits, likes, shares, comments, reviews, ratings, tweets, photos, videos etc. Apart from these web services, big data is also generated by countless sensors all around the globe to gather valuable information about respective fields. For example, closed circuit cameras recording is a good source of big data. In a nutshell, every electronic device, Internet services, embedded system and sensor the globe produce and transmit data and big data is being generated by these multiple sources at an alarming velocity, volume and variety.

For researchers, however, big data brings new sets of challenges like how to efficiently and effectively handle big data in commodity computers, how to apply conventional algorithms, how to properly manage big data in distributed setting etc. Specially, in field of machine leaning and Pattern recognition, big data poses a threat, because the conventional algorithms were designed solely to fulfil its purpose where entire dataset can fit in memory. Distributed setting was not taken into consideration. In nutshell, to extract any meaningful insight from this big data, we need powerful tools to analyse it, elastic frameworks to cope up with its sheer volume and most importantly we have to rethink and redesign existing algorithms which are most suitable for smaller sized dataset and do not take advantage of clusters.

In this chapter we will discuss the present issues that we are going to take into consideration in our research. We will indicate the challenges we targeted and in later chapters we will give our proposals in order to minimize these challenges in data analysis.

4.1 Tall and Wide Data

In general, the width of data means the number of features that each data sample is comprised of. In present world, this features count is increasing at large scale. The feature count or width of data sample can also be named as "Dimension of Data". One of the most effective use of PCA is dimensionality reduction. Many machine learning algorithms are not capable of handling data with large dimension. Therefore, we can use PCA to reduce the width of data then fit them in some kind of machine learning model. This present some kind of new challenge. The present PCA algorithms are not that much capable of handle very wide data that is tall also. Therefore before making the data suitable for machine learning models by dimensionality reduction, the PPCA algorithm has to deal with the curse of tall and wide data. The present implementation of *sPCA* done good number of optimization to make efficient use of memory. Still it is not capable of handling tall and wide data. The memory overflow occurs while computing the intermediate results.

4.2 The Blessings and Curses of Geo Distributed Data

Geo Distributed Data has both positive and negative sides. In this section we will show some of them.

4.2.1 The Bright Sides

Assurance of Faster Data Access

As data stays closer to end users, the users can easily access the data with lower latency.

Protection of National Privacy

Now data contains some kind national privacy. Moreover no nation will want the data of its people to cross the national boundary. As a result there are regulations on na

Sparsity of Data Centers

The Data centers are not places on the same regions. Multiple Data centers are placed in different place.

Development in Business Sector

As different giant companies establishing there data center in different countries, so the business is growing.

4.2.2 The Dark Sides

- New Challenges in Data Analysis
- Unavailability of Global View
- Break Down of Data Consistency

Chapter 5

Preliminary Work on Regression Analysis

The field of geo-distributed data analysis is quite vast. A detailed study of these aspects of the distributed approach of data analysis is beyond the scope of our tasks during the first few days. As a part of learning how to work on data analytic what we wanted to introduce is a problem specific approach. We were interested in doing "Regression Analysis" on data located at different geographical areas all around the world. Primarily we have worked out with only two variants of regression analysis "Linear Regression" and "Polynomial Regression". So we can say the other variants as well as other problem oriented approaches were not be covered in this chapter.

5.1 Contribution

Summarizing, our main contributions are:

- We propose an approach of regression analysis that is deals with geo-distributed data and provide an in-depth study of the relative merits against centralized approach.
- We propose a system that builds upon Apache Hadoop MapReduce [37] framework and extends their functionality to support multi data center learning applications.
- We propose an approach that will minimize the number of transfers among data centers thus minimize the inter data center bandwidth utilization.
- We propose an approach that is a combination of the two concepts *Data Parallelism* [38] and *Model Parallelism* [38].
- We present experimental results implemented on *Amazon Web Services (AWS)*.

5.2 Problem Specification

We were interested in building a regression model on geographically distributed data. We had the intention of using least square regression analysis. In order to facilitate the study, we formalize the problem in two dimensions:

1. We will make our assumptions about the data, its type, size and distribution.
2. We limit the regression analysis to 2nd order only. That means we would try only "Linear Regression" and "Polynomial Regression".

So we can see that regression analysis on higher order and other classes of regressions (eg. "Logistic Regression") will not be covered in this paper.

5.2.1 Assumptions on Data

To keep the analysis simple we used data only in text format. There may be multiple text files which will be considered as input files to our program. every text file contains multiple lines and each of these lines will be tokenized and the information will be extracted.

5.2.2 Distribution of Data

In order to run our program on distributed data we used cloud computing facilities of *Amazon Web Services (AWS)*. We created two different data centers in two different geographic location, "North Virginia" and "Malaysia". Every data center was consisted of two *Linux* instances. Among the two instances of a data center we named one of them as *NameNode* and other one as *DataNode*. The *NameNode* works as a master node of its corresponding data center. Full data of a particular data center is further distributed in its *NameNode* and *DataNode*. Any kind of jobs on the the data of a data center can also be split and distributed among its instances.

5.3 Approach

We created *Hadoop MapReduce* cluster on *Amazon AWS*. Our implementation needs to obtain resources like CPU cores, Memory during the partial computation on data in a particular data center in a uniform basis. Such resources are managed by a resource manager in the *Hadoop* cluster system. Further, *Apache Hadoop YARN*'s new federation feature allows us to view multiple data centers as a single one. Data on a data center is organized by *Haddop* File System *HDFS*.

5.3.1 Learning Process

For running our regression computation on geographically distributed data we mainly focused on two concepts:

- **Data Parallelism** It is the main idea of distributed data. Its can be seen as application of a single instruction to multiple data items. It focuses on distributing the data across different nodes, which operate on the data in parallel. As an example we can mention the array operations. In case of summing the data of an array, we can make multiple partition of the array and then apply summation formula on each partition. The aggregated result will give us the final summation. In our distributed data organization, data are located in different location while we need to make the same computation on them. This can be achieved by making partial computation on each data partition and aggregate them at the end.
- **Model Parallelism** In case of model parallelism the system gives every processor the same data but applies a different model to it. Model parallelism finds its applications in deep learning. As we can see that data centers are generally composed of multiple machines, we can use that resource to speed up the computation. If any learning process consists of multiple types of computation then we can create different model and place them in different machines. Therefore, at the time of making partial learning from any data center, each machine can do its model task on a parallel basis and the whole result can be obtained at a lower time period.

The first innovation in our approach is that we intend to make a mixed use of both *Data Parallelism* and *Model Parallelism*.

- Data that are distributed in different data centers located at different geographic location resemble the *Data Parallelism* concept.
- Then in case of a single data center, every machines of that data center has a shared view on the data because of the *HDFS* file system. In order to compute n parameters, if the data center has n machines then we can distribute the different computation in different machines. That means every machine will work on separate models in parallel. This gives the use of model parallelism.

The parameter to passed among the data centers can be determined from the equations or regression. For example of we consider regression of second degree, it can be expressed as:

$$y = y_{model} + \epsilon$$
$$y_{model} = ax^2 + bx + c$$

where,

$$y = original\ value$$
$$y_{model} = model\ value$$
$$\epsilon = error$$
$$\epsilon = y - y_{model}$$

If we intend to use least square regression model then we have to minimize the total squared error

$$\begin{aligned} S &= \sum \epsilon^2 \\ &= \sum (y - y_{model})^2 \end{aligned}$$

In the process of minimizing S we get the necessary equation to find the co-efficients in matrix form:

$$\begin{bmatrix} \sum x_i^4 & \sum x_i^3 & \sum x_i^2 \\ \sum x_i^3 & \sum x_i^2 & \sum x_i \\ \sum x_i^2 & \sum x_i & n \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} \sum x_i^2 y \\ \sum x_i y_i \\ \sum y_i \end{bmatrix}$$

Here we have to compute 7 parameters that appear in the matrix form. Every data center will compute these parameters on its own data. Then they will be sent to the initiating data center. At the initiating data center all the parameter will be merged to get the final model.

5.3.2 Algorithm

The whole process is divided into two parts. First of all, every data center will compute parameters by executing computation on their corresponding partial data. This task will be handled by the child processes. And the initiating data center i.e. master data center will run the parent process. which will receive the partial computational result and merge them to built a complete regression model.

Algorithm 2 Geo-Distributed Regression Analysis

Input: Data located at different geographic location
Output: Regression model
Parent Process
select an arbitrary data center as the initiating one, *root*
while end of "node list" has not encountered **do**
  pick a data center, *node*
  create a parallel executable child process with the parameters *root* and *node*
end while
$available = false$
while $available = false$ **do**
  if every partial computational files from every data centers are available **then**
    $available = true$
  end if
end while
merge all the partial computational files into a single one
create final regression model using the merged files

Algorithm 3 Geo-Distributed Regression Analysis

Input: Data located in corresponding local machines
Input: Information of 2 nodes
Output: Partial Parameter
Child Process
set the input and output locations of *Hadoop* File System
run the Partial Regression Computation program
if root and node are the same **then**
  the $node$ is the initiating data center $root$ itself
  rename the output files with information of $root$
  move the renamed files in the $CommonContainer$ location of $root$
else
  the $node$ is a non initiating data center
  rename the output files with information of $node$
  send the renamed files in the $CommonContainer$ location of $root$
end if

From the algorithm, we can see that there will be two types of communication during the learning task:

- **Global Communication** Every data center has multiple instances of machines. Among them only one will communicate with other data centers. Such machine will be considered as a master node. There will be message passing among the master nodes of data centers. Whenever a learning process is to be initialized one of the master nodes will issue command to each master nodes to start their partial computation. Again after the partial computation of each data centers the parameters will be passed towards the initializing data center's master node. In this way the master nodes of every data center will create a Global Communication Group among them.
- **Local Communication** Whenever a data center's master node receives the command of partial computation, it will communicate with its other machines i.e. slave nodes. The computation task will be distributed among all the machines of that data center in order to exploit the advantage of *Model Parallelism*. We can say that each machine of a data center will work on one or some specific job. At the completion of each job each separate machine will write its result in a common container of the master node from where all the partial results will be transferred to initialing data center's master node.

The communication group are shown in Figure 5.1

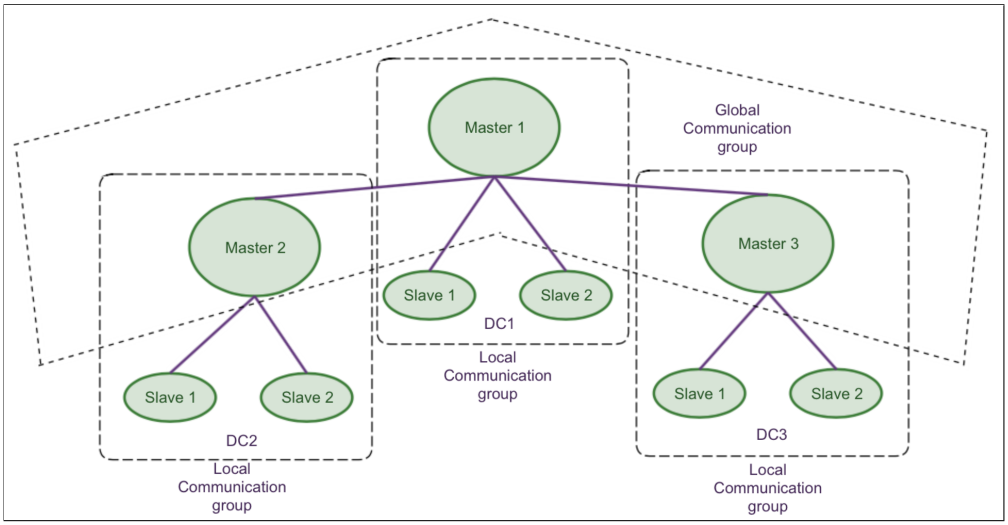


Figure 5.1: Communication Groups

Figure 5.2 shows how the algorithms works. Here direction of the arrows shows the transfer direction.

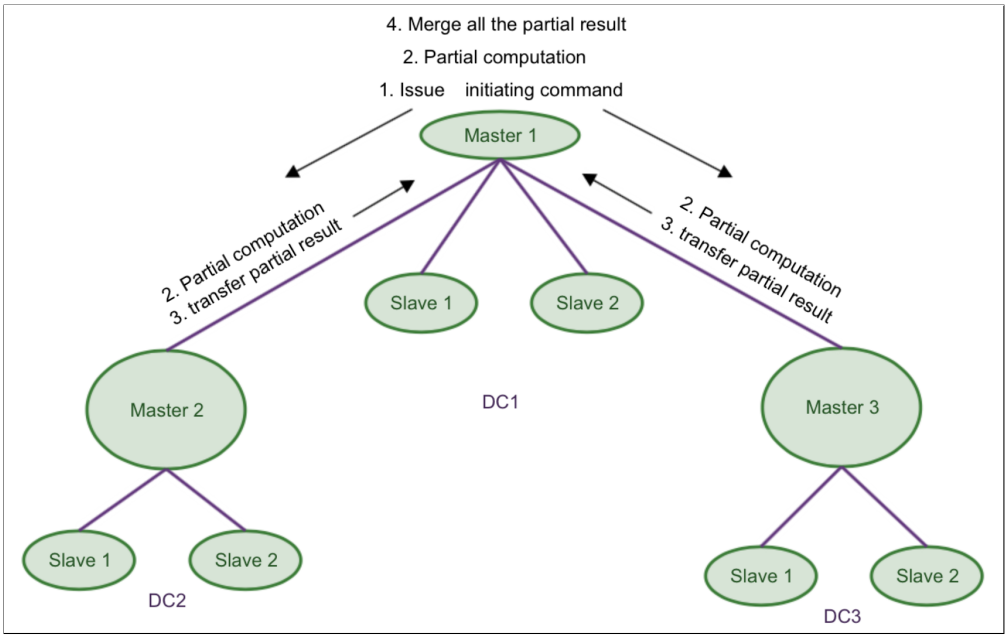


Figure 5.2: Graphical Interpretation of Learning Algorithm

5.4 Our Implementation

For the main program we used *Hadoop MapReduce* framework. We used *Java* as programming language. For inter data center communication we used *Secure Shell (SSH)* Protocol. For some extra works we had to use *C++* programming language also.

5.4.1 How the Distributed Algorithm Works

A shell script initializes the whole learning process. Master node of every data center will contain this program file. Every data center will have the information of other data centers where data reside in distributed fashion. This program will read every other data center's information and with this information it will start another child program along with it's own information.

The child program receives information of all the data centers one by one. Then it will communicate with that data center and issue a command to start it's internal computation for parameters to build a complete regression model.

The child process will match the information of two data centers passed to it. If it is the initiating data center, then it will start some master task. Other wise slave tasks will be initiated.

At first environment variables associated with *Hadoop MapReduce* will be set up. At this point the main program of computing partial results will be executed on every data centers. The program is based on mapping input data into specific intermediate outputs and then reducing them. For example, while running the computation model of $\sum x^k$, the map function will map every x into x^k and send to the reducer. The corresponding reducer will sum all those x^k to

generate $\sum x^k$.

When the partial computational output is available in initiating data center, this program will rename the output files with appropriate names including corresponding data center information and move them into a common container.

The main difference of non initiating data centers is rather than moving the renamed files this program will send those files to the common container.

When all the partial computations are completed, the margin process will combine all the files of this container.

5.4.2 Runtime Analysis

If the size of data is n, i.e. there are n rows in every sample input then in the *MapReduce* program, the number of maps for each parameter model will be $O(n)$. If the number of parameters to be generated is p and number of machines is $\frac{p}{m}$ then the number of maps will be $O(mn)$
If the reducer of a parameter merges every two intermediate outputs of corresponding mapper then we can minimize the number of reducers to $O(\log_2 n)$. Therefore if the number of machines is $\frac{p}{m}$, the number of reducers will be $O(m \log_2 n)$..

5.5 Experimental Findings

In case of finding any parameter, while any job is submitted it takes few time or its initial setup. For analyzing the result we ignored that setup time by finding it empirically. First of all we show the time vs data size graph. From Figure 5.3, we can see that the time grows almost linearly with the size of data.

A comparison between centralized approach and our distributed approach is shown in Figure 5.4. We can easily see that time required for centralized approach is growing at a pretty high rate than that of the distributed approach.

As we were making an use of *Model Parallelism*, the number of instances in a data center plays a big role in the computational runtime. We can demonstrate it by an example.

- In case we have n models to run and have $\frac{n}{k}$ instances in a data center. Then every instance will be assigned to run k models. It means these k models will run at a sequential basis. Obviously the number k is preferable to be as low as possible.

- The best result can be found with $k = 1$. While the number of instances are equal to the number of models then every model can be run in parallel. Then the total runtime will be

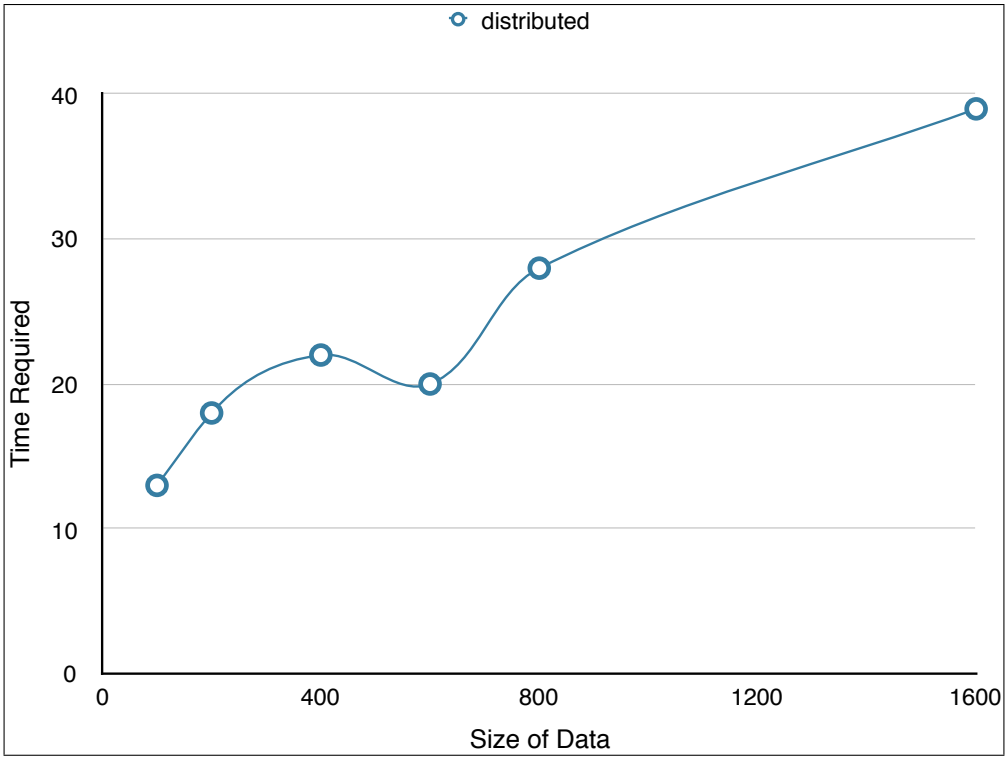


Figure 5.3: Time Required vs Data Size graph of our Distributed Approach

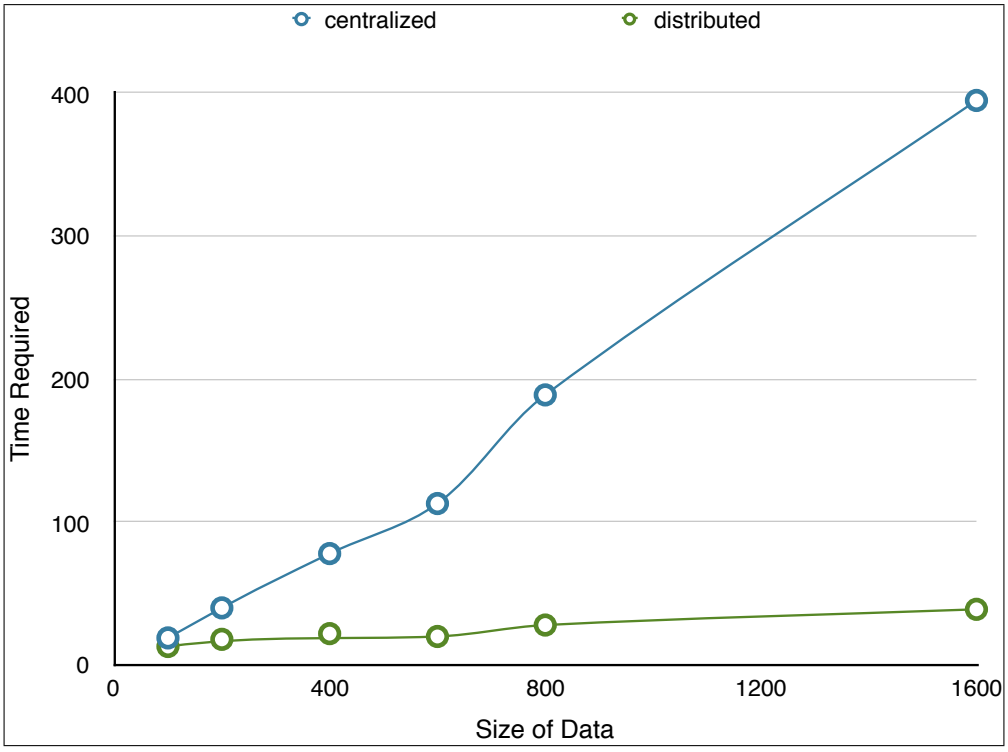


Figure 5.4: Comparison graph between centralized and distributed approach

the time of the worst time taken model.

As we were using free tires of 750 hours in *Amazon Web Services*, we did not have the scope to use as more instances as possible. In fact, we could only use 3 instances at most for every data center. Still it showed a significant variance in runtime while running on different number of instances. In Figure 5.5, we can see that the run time decreases as the number of instances increases.

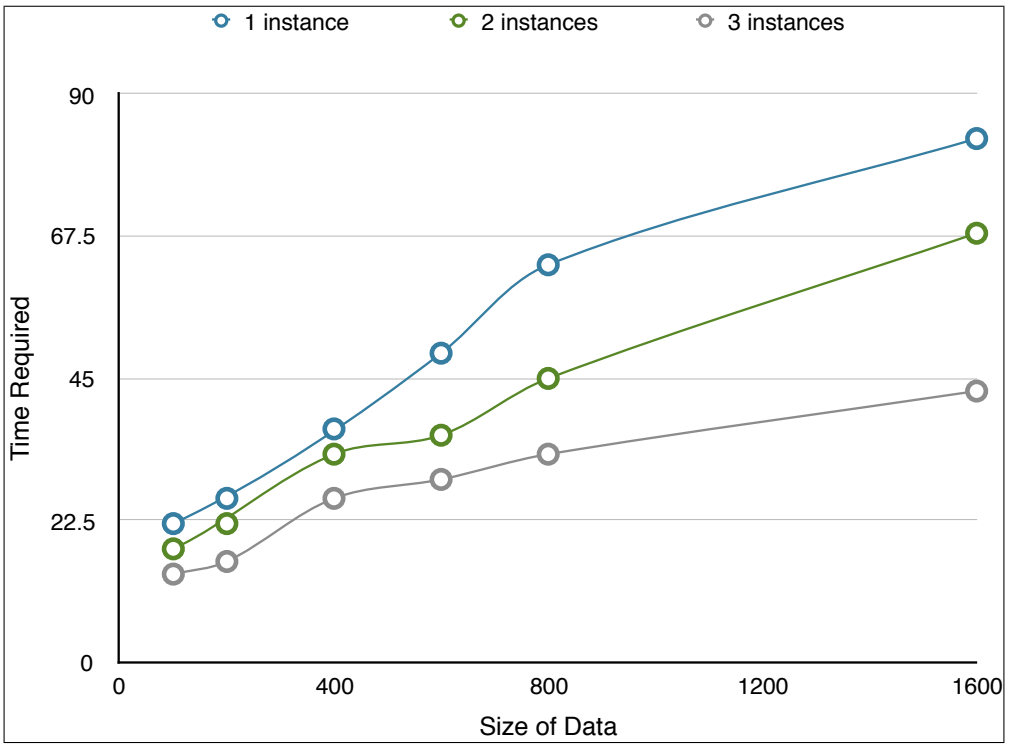


Figure 5.5: Comparison graph based on number of instances

5.6 Limitations

We used a general approach to create the regression models for both linear and polynomial regression. Each data center does its portion of computation on its own data. We applied the approach on mainly text input files where each file consisted of multi line texts. For each line we extracted 2 parameter as x and y. Here we can see that we are finding sums such that $\sum x^2$, $\sum x^3$, $\sum y^2$, $\sum y^3$ etc. To be specific in order to create a linear regression model we need to calculate sums of exponents of x and it was up to $\sum x^2$ then for polynomial it was up to $\sum x^4$. In general it can be observed that for a regression analysis of nth order we need to calculate up to $\sum x^{2n}$ and have to send $3n + 2$ numbers of such summation values to a central data center. So we can see that using higher order regression analysis it will become practically impossible to use this approach.

More problems arise while x's carry large values. In case of polynomial regression if in general every data center has one billion 10^9 line text input in data files and the average value of x is in range 10^3, then the system will have to handle sum values in the range of 10^{21}. A different approach of handling such big numbers should be followed to make the approach more useful.

The time required for the computations of parameters are not same. Primarily, in case of using *Model Parallelism* if we have exactly same number of instances and models then we assume to have the best runtime. But all the models dont take the same time. As a result the load of running models can be further distributed among idle instances. Whenever any low time consuming model is finished in any instance, it becomes idle and any high time consuming model can be split and a portion can be assign to run in such idle instances.

Chapter 6

Our Proposed Approach

As discussed before, we are interested in presenting an approach that will resolve the memory limitation problem and hence give an systematic way to run *PPCA* on geographically distributed data.

6.1 Handling Tall and Wide Big Data

Our first concern was to make commodity computers capable of performing PPCA on such big data which has very large sample count as well as very high dimension i.e. very large feature count. In this respect we will follow the approach from [22] with some necessary improvements. To have a distributed settings we will take advantage of cluster computing. In case of very wide data the size of principal subspace W is going to be very big. Therefore, we have to treat the principal subspace W as big data also. Performing PPCA in the conventional approach on tall and wide big data will result in the overflow of memory. Therefore, we have to make some kind of improvements to make the approach capable of handling tall and wide big data in case of performing PPCA overcoming the memory scarcity. The steps of performing PPCA in a single cluster is as follows:

6.1.1 Step 1 - Partition of Principal Subspace W

For data with the dimension $N \times D$ we will get a principal subspace of dimension $D \times d$ where d denotes the number of principal components we want. Working with full horizontal D dimension of W will result in memory overflow. Therefore, we will make n horizontal partition of W and create $W_1, W_2, \ldots, W_n$. As a result each stage of creating W will consists of n smaller stages of creating W_i.

6.1.2 Step 2 - Data Partition

As we are partitioning the horizontal D dimension of principal subspace W, we are indirectly make vertical partition of our data of $N \times D$. At each iteration of generating W, we have to work on a specific horizontal segment of it. Therefore, we have to work with only that segment of dimension of our data. In that sense we can say that we are creating vertical partition of our main data matrix. It might be a case that data is pretty large that even a distributed setting with multiple machines might not be capable of performing the PPCA task by keeping the full data in memory. In such case our approach can be memory efficient by loading only a single segment of data at each iteration. This will help in memory intense tasks.

6.1.3 Step 3 - Expectation Step and Omission of Noise Model

From Algorithm 13 we can see that computation of $\boldsymbol{\sigma^2}$ involves X which is going to a $N \times d$ matrix. Therefore, passing such big data will result in communication bottleneck. For simplicity of computation and minimizing the inter data cluster communication, we will omit the noise calculation.

On the other hand, in order to take advantage from segmented computation, we have to make some changes in the Expectation and Maximization steps of the main **EM Algorithm** of PPCA. Omitting the noise model and from Equation (3.12), (3.13) and Algorithm 13, the E-Steps are:

$$M = W^T * W \tag{6.1}$$

$$X = Y_c * W * M^{-1} \tag{6.2}$$

$$XtX = X^T * X \tag{6.3}$$

$$YtX = Y_c^T * X \tag{6.4}$$

In our approach M is not going to be produces at the start of the process as ot depends on W which will be segmented in our system. Therefore we are going to produce multiple number of M's and accumulate them to produce the final M. The steps equivalent to Equation 6.1:

$$M_i = W_i^T * W_i \tag{6.5}$$

$$M = \sum_{i=1}^{n} M_i \tag{6.6}$$

Similar thing will happen for producing X of Equation 6.2:

$$X_i = Y_{ci} * W_i \tag{6.7}$$

$$X = \sum_{i=1}^{n} X_i \tag{6.8}$$

Steps to produce XtX:

$$\begin{aligned} X &= X * M^{-1} \\ XtX &= X^T * X \\ &= (X * M^{-1})^T * (X * M^{-1}) \\ &= M^{-1^T} * X^T * X * M^{-1} \\ &= (M^{-1})^T * (X^T * X) * (M^{-1}) \end{aligned}$$

Similarly we will produce YtX

$$\begin{aligned} X &= X * M^{-1} \\ (YtX)_i &= Y_{ci}^T * X \\ &= (Y_{ci}^T * X) * M^{-1} \end{aligned}$$

Step 4 - Maximization Step

As we have mentioned earlier we are omitting the calculation of variance $\boldsymbol{\sigma^2}$. Therefore, our maximization step gets limited to the only computation of new W. As we are generating W in a segmented approach, at iteration i we are going to generate W_i as follows:

$$Wi = (YtX)_i * XtX^{-1}$$

Here $(YtX)_i$ is the YtX generated from Y for the range of dimension corresponding to i.

6.2 Flow Graph and IO Operations

Figure 6.1 shows the flow of the algorithm of handling tall and wide big data in a single cluster. The algorithm will start from a random state. It will randomly generate initial segments $W_1, W_2, \ldots, W_n$. Then save them in File System. At each round the algorithm will load a particular segment W_i and with the corresponding dimensions of data Y_i it will generate M_i and X_i. after n iterations complete X and M will be generated. Then using X, M^{-1} and M^{-1^T}, it

will generate XtX. At the same time using Y_i, X and M^{-1}, it will generate particular $(YtX)_i$. Then using XtX and $(YtX)_i$, the algorithm will generate new W_i and store it in File System and then call the accumulation process to start for that particaulr W_i. 4 is the Algorithm to handle tall and wide biog data in a single cluster while

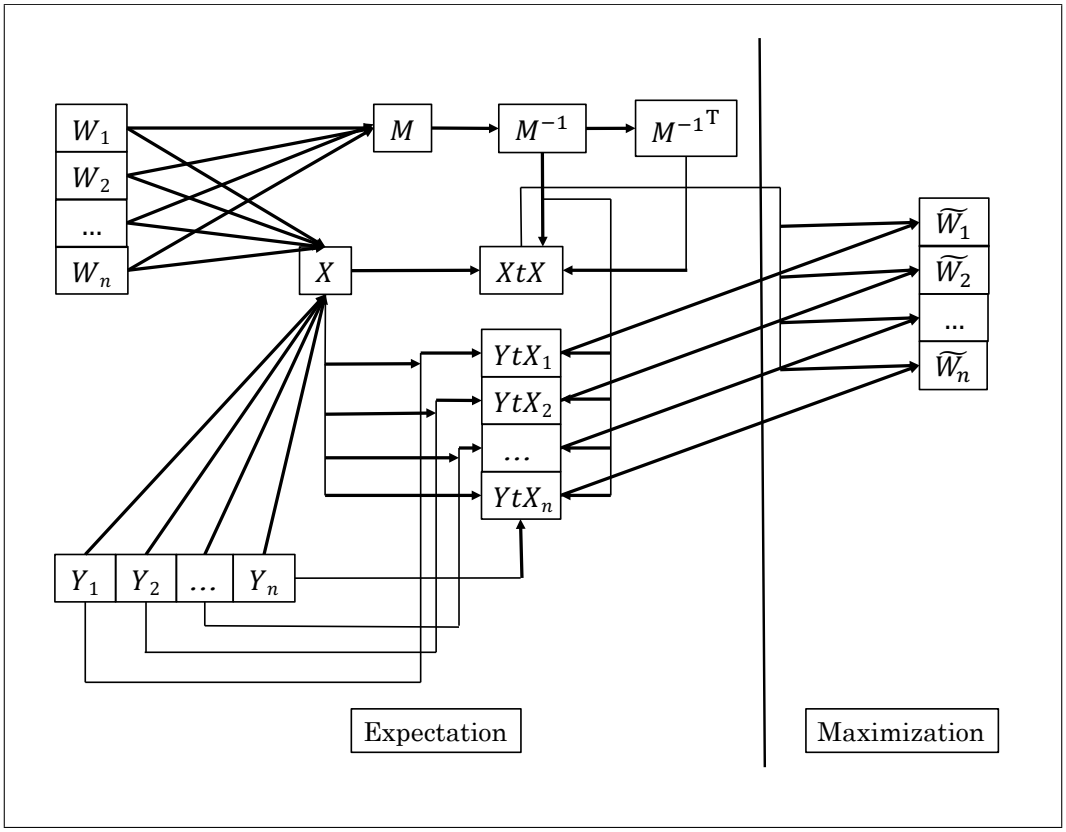


Figure 6.1: Flow Graph of Algorithm 4

6.3 Accumulation of partial results from geographically distributed clusters

As we mentioned earlier our data will be geographically distributed. That means we are not going to have any global view of data and no raw data passing is allowed to preserve national data sovereignty. Therefore, the partially computed results have to accumulated to produce the final result. We had the intention to minimize (a) the volume of data to be transmitted throughout the whole process and (b) accumulate the partial results in a way that will ensure the minimum accumulation time. The former was handled in the previous subsection. In The later part we are going to be use some graph theory properties to achieve what we desire.

At each iteration of PPCA we are generating a segment of our principal subspace W. An arbitrary segment can can be denoted as W_i. Whenever one such segment is newly generated, each

Algorithm 4 PPCA on Tall and Wide Big Data

createMyInfo()
$myInfo$ = ***readFile***$(myInfo)$
Let $Range[1.....partitionCount + 1]$ be an array
for i from 1 to $partitionCount$ **do**
  $Range[i] = (i - 1) \times D \div partitionCount$
end for
for i from 1 to $partitionCount$ **do**
  $start$ = $Range[i]$
  end = $Range[i + 1]$
  W_i =***GenerateRandomW***$(start, end, d)$
  saveWInStorage(W_i)
end for
$Stop\_Condition$ = $False$
while (!$(Stop\_Condition)$) **do**
  $M = null$
  $X = null$
  for i from 1 to $partitionCount$ **do**
    $start$ = $Range[i]$
    end = $Range[i + 1]$
    W_i = ***loadWFromStorage***$(start, end)$
    $M_{new} = W_i^T \times W_i$
    $M = M$ + M_{new}
    $X_m = Y_m \times W_i$
    SegementedXJob$(X, Y, X_m, W_i, start, end)$
  end for
  $invM$ =***invert***(M)
  $YtX = null$
  $XtX = null$
  for i from 1 to $partitionCount$ **do**
    s = $Range[i]$
    e = $Range[i + 1]$
    generateYtXandXtX$(YtX, XtX, X, Y, Y_m, i, s, e)$
    $YtX = YtX \times invM$
    $XtX = invM^T \times XtX \times invM$
    $Wi = YtX \times$ ***invert***(XtX)
    startAccumulation$(myInfo, i)$
  end for
end while

data cluster will call the **Accumulation** procedure from Algorithm 6. The full accumulation process that we are proposing can be presented as follows:

6.3.1 Step 1: Generating The Initial Graph

The data clusters that are geographically distributed are connected by high or low bandwidth connection based on their location. According to [33] and [34]:

- Clusters that are located at the same region generally have connection of bandwidth in the range above $100gbps$
- Clusters that are located at nearby regions generally have connection of bandwidth in the range of 10-$100gbps$
- Clusters that are located at different and distanced regions generally have connection of bandwidth in the range of 1-$10gbps$

At the start of the full PPCA process, every data cluster will check the bandwidth of the connections of every other clusters it is connected with and generate a graph. The full process will results in a complete graph if there is an interconnection between every two data clusters. In this graph every node v_i will represent a data cluster DC_i while edge $e_{ij}(cost)$ between nodes v_i and v_j denotes that DC_i and DC_j are connected at a b/w using which certain amount of data needs $cost$ amount of time to transmit among the two vertices. The Figure 6.2 shows such a generated graph with 10 data clusters.

6.3.2 Step 2: Generating MST

Using the graph generated in *step 1*, every node will generate a *Minimum Spanning Tree* to make a connected component with the least cost using time required to transmit data as the cost of the tree edges. Figure 6.3 shows the MST generated from graph from Figure 6.2.

6.3.3 Step 3: Sub Tree Generation for Parallel Accumulation

We are going to generate two sub trees from the MST generated in *Step 2* according to Algorithm 5. We will intended to choose the maximum time consuming edge e_m and make a sub tree rooted at each of the end vertices of that edge. This will be done to make the process to

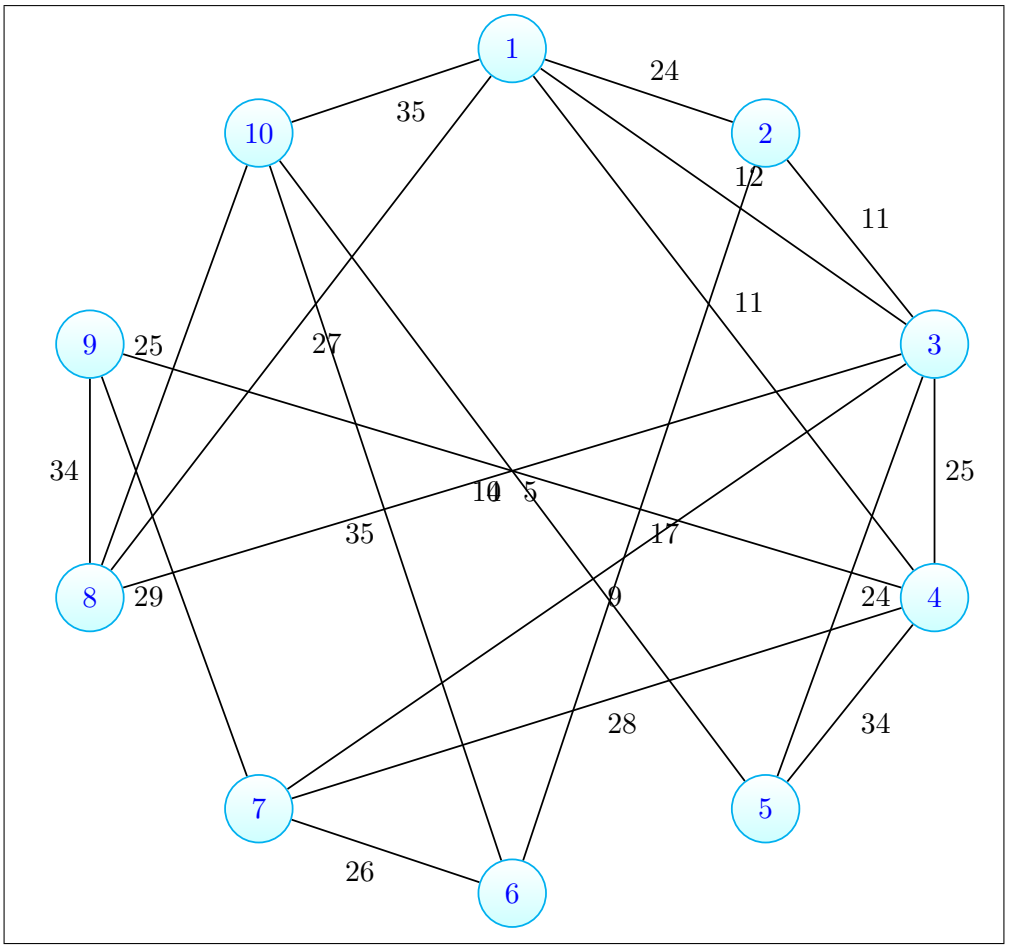


Figure 6.2: Generated Graph G According to Step-1

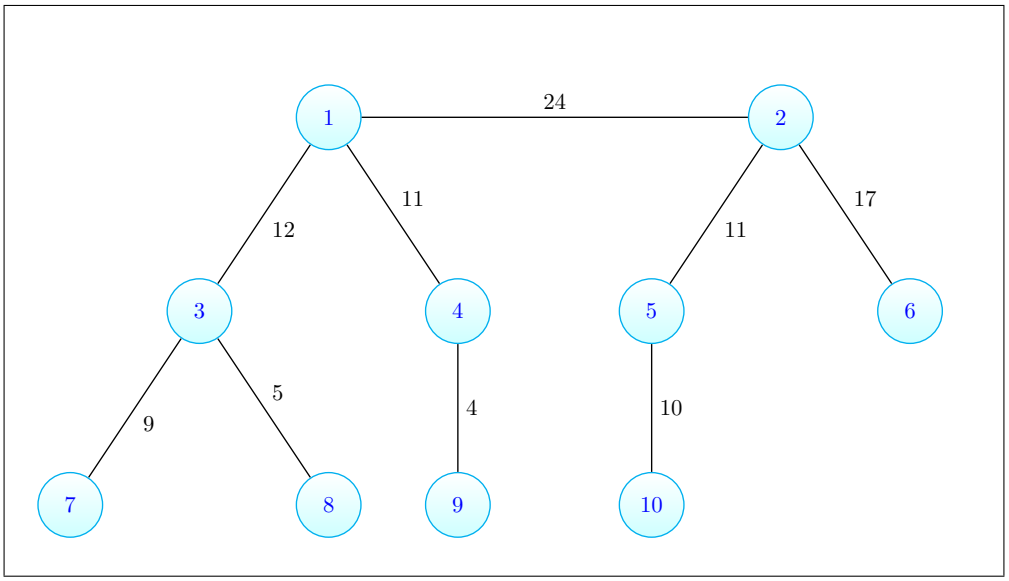


Figure 6.3: Generated MST According to Step-1 from Graph G

use the maximum time consuming link only once while accumulating data. What we are trying to achieve is that all the data cluster nodes at each of the sub trees now can perform the accumulation task in parallel. Accumulation will be done in bottom up direction. Therefore two accumulated results will available at the two root data nodes. At that point a single exchange will results in having the final result at each of the root node of the two sub trees.

From this sub tree generation with using the node information, every data node will learn the information of its child nodes and parent nodes. During accumulation it will accumulate results from its child data nodes and notify the completion of it accumulation to its parent node.

What we keep in mind that this subtree formulation method might create some kind of unbalanced subtree. Figure 6.4 shows such an unbalanced sub tree where $subtree1$ has a total cost of 51 and $subtree1$ has a total cost of 21.

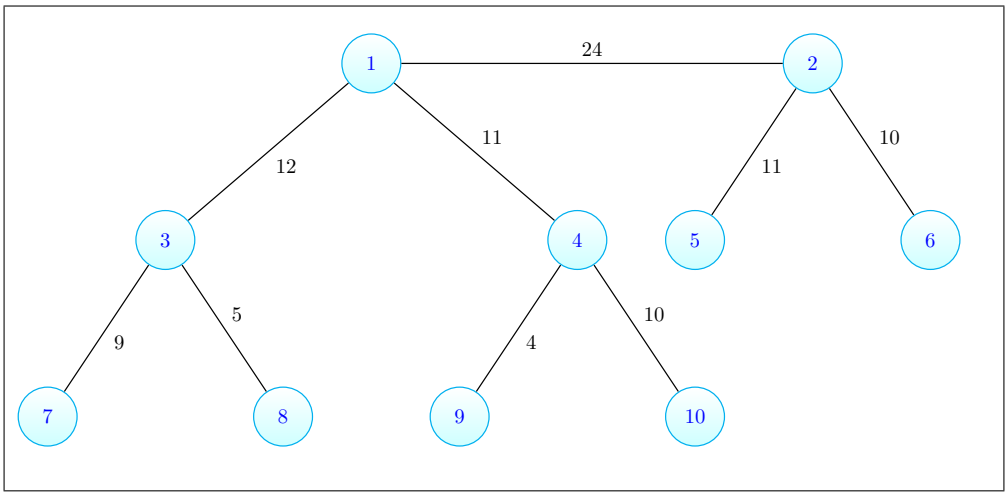


Figure 6.4: Unbalanced Sub Tree

To overcome such condition, we will then make a trade off between choosing the highest time consuming link and the balanced cost of the sub trees. To make the sub tree costs more or less balanced we will try to take the next maximum time consuming link and so on.

6.3.4 Step 4: Redistribution of Final Accumulated Result

Whenever each of the root nodes completes the accumulation of its own sub tree, it will wait for the other root to complete. After that it will access the partially accumulated result from the other root node and complete the full computation. At this stage it will distribute the final result by pushing it through the links connecting to its neighbours. As there will be dedicated links, the result can be sent to each of the neighbours at the same time. Whenever ant node in a sub tree receives the final result it will also redistribute it to its child nodes in the tree configuration. The job will end with the leaf data nodes receiving the final accumulated result.

Algorithm 5 createSubTree

1: **Input:** V_m, Vertex Set of MST
2: **Input:** E_m, Edge Set of MST
3: **Output:** $\{subTree1, subTree2\}$, Two Subtrees
4: sort E_m according to increasing b/w
5: $done$=$Fasle$
6: **while** !$done$ **do**
7: e = *nextMinimumBWEdge*(E_m)
8: $v1$ = $e.startVertex$
9: $v2$ = $e.endVertex$
10: $subTree1$ = *subTree*($v1$)
11: $subTree2$ = *subTree*($v2$)
12: $cost1$ = *cost*($subTree1$)
13: $cost2$ = *cost*($subTree2$)
14: **if** (*max*($cost1, cost2$)/*min*($cost1, cost2$)≤ 1.5) **then**
15: $done = True$
16: **end if**
17: **end while**
18: **return** $\{subTree1, subTree2\}$

The Algorithm 6 is for accumulating the partial result. It will be called from every data nodes (leaf or non leaf) and according to the node information generated by Algorithm 7 will communicate with other nodes in the communication tree.

6.4 Communication Groups

From the preceding subsections we can see that there will be two types of communication:

- **Local Communication** This type of communication persists during the handling of tall and wide data in a single data cluster. Here all the slave data nodes of a single data cluster will work on a specific segment of the data matrix Y. While the single segment of W will be available in every data node at a time. The master node will handle the partitioning of the data matrix while as well as the scheduling tasks. The slave nodes will work on their specific data segment and generate partial results that will be passed to the master node in order to generate the aggregated one.

- **Global Communication** During the accumulation processes since W_i's will be saved in the File System that will be under control of the master node in a data cluster only the master data nodes of different data clusters will communicate among themselves. This communication group is referred to as global communication group.

The communication group are shown in Figure 6.5

Algorithm 6 startAccumulation

Input: $myInfo$
Input: $indexOfW$
W = ***loadW(***$indexOfW$***)***
for (each $childNode$ in $myInfo.child()$) **do**
  if ($childNode.WisReady(indexOfW)$) **then**
    W_{child} = ***getWFromNode(***$childNode$***)***
    W = W+W_{child}
  end if
end for
notifyParent($myInfo.parent$***)***
if ($myInfo.rootNode$) **then**
  Wait Until Other Root Node Data is Ready
  $W_{otherRoot}$ = ***getWFromNode(***$myInfo.otherRootNode$***)***
  W = W+$W_{otherRoot}$
  Announce $doneW(indexOfW)$ as $True$
  for (each $childNode$ in $myInfo.child()$) **do**
    transmit(W***)***
    transmit($doneW(indexOfW)$***)***
  end for
else
  while (!$doneW(indexOfW)$) **do**
    wait()
  end while
  for (each $childNode$ in $myInfo.child()$) **do**
    transmit(W***)***
    transmit($doneW(indexOfW)$***)***
  end for
end if

Algorithm 7 createMyInfo

Input: V : Set of Clusters as Vertices
Input: E : Set of Weighted Edges; b/w as Link Weights
Input: $G = \{V, E\}$ Clusters' Distribution Graph
$MST(V_m, E_m)$ = ***createMST(***G***)***
$\{subTree1, subTree2\}$ = **createSubTree(**V_m, E_m**)**
$myInfo$ = ***createMyInfo(***$subTree1, subTree2, myID$***)***
saveMyInfo($myInfo$***)***

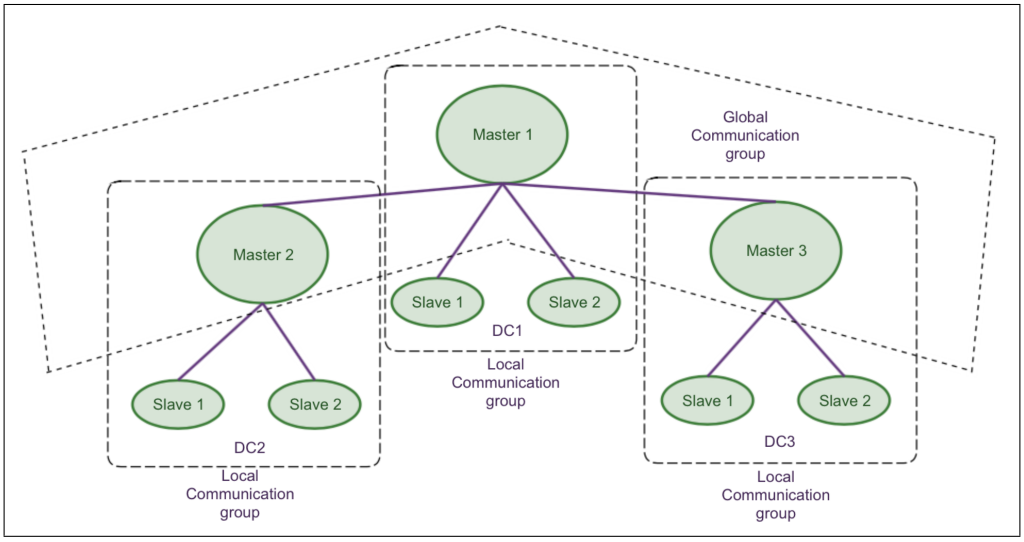


Figure 6.5: Communication Groups in TallnWide and Accumulation

Chapter 7

Properties of our Approach

In this section, we analyze the computational and communication complexity of our proposed algorithm and derive some theoretical properties.

7.1 Computational Complexity

We will discuss the computational complexity for Algorithm 4 that do the task of computing principal subspace W pf the input data Y. For computational complexity analysis, we will consider a spark-like [35] distributed setting where in a data cluster each of the data nodes has a portion of full data and intermediate results are stored in-core. Specifically if there are c data nodes in a single cluster and each of them have approximately $N_i \times D$ data matrix Y_i, then we can write:

$$Y = \sum_{i=1}^{c} Y_i$$

We assume the partition count of W to be p. if every segment of W is of size $D_i \times d$ where d is number of principal components to be computed. Then we can write:

$$W = \sum_{i=1}^{p} W_i$$

Now our Algorithm 4 has three different parts.

From line 7 to 12 a **For Loop** is generating initial random W_i's. Here the running time is $\mathcal{O}(pD_id)$

From line 17 to 24 another **For Loop** is there to compute X of Equation 6.5 and 6.6. Here tasks of line 20-24 will have a time complexity of $\mathcal{O}(D_id^2 + d^2 + d + d + N_iD_id)$. That makes the aggregated runtime of this for loop $\mathcal{O}(pN_iD_id)$.

From line 29 to 37 another **For Loop** is generating XtX and YtX, fining at generating new W_i. Therefore line 32-35 will have a time complexity of $\mathcal{O}(N_i d^2 + D_i N_i d + D_i d^2 + d^4 + D_i d^2)$. That makes the aggregated runtime of this for loop $\mathcal{O}(pN_i D_i d)$.

Therefore, if **While Loop** of line 14 runs for r rounds then the total run time is going to be $\mathcal{O}(2rpN_i D_i d)$.

7.2 Communication Complexity

The approach is highly communication intensive. At the end of generating each new segment of principal subspace W each data cluster will call Algorithm 6 as a background process. This will make a data cluster to accumulate from its child nodes and give a notification to its parent. Each accumulation demands the transmission of a $D_i \times d$ matrix where D_i varies with the number of partition is done on W. As partition information i.e. D_i is available at the start of the process the communication tree generation can be done based on the available bandwidth between any two data clusters and the time required to transmit data of that size. For simplicity we assume the availability of dedicate bandwidth between any two data clusters.Under these assumptions, the time needed to make the accumulated result of a subtree available at the root node of that subtree is equal to the maximum aggregated time of any path from root to leaf node. If such paths are P_1 and P_2 of $subTree1$ and $subTree2$ respectively and maximum time consuming link time is max_time, then the time of communication will be

$$2 * max\_time + 2 * max(time(P_1), time(P_2))$$

The 2 is added as accumulated data will be redistributed in the same paths. In Figure 7.1 the maximum time consumtioon link has a time of 27 units the gray shaded paths has the maximum time consumption of 46 and 31 units. Therefore the communication time of accumulating one segement of W will be

$$2 * 27 + 2 * max(46, 31) = 146\ units$$

7.3 Stop Condition

From Algorithm 4 we can see that the **While** loop of line 14 will continue until the stop condition being true. While the algorithm will start we will set a tolerance rate. While the reconstruction error comes within the tolerance rate the algorithm will be considered to reach the convergence and eventually it will stop further iteration. The error checking formula is given by [32]: $e = ||Y - X * W^{-1}||_1$ However, since the sampling rate and size of data matrix will

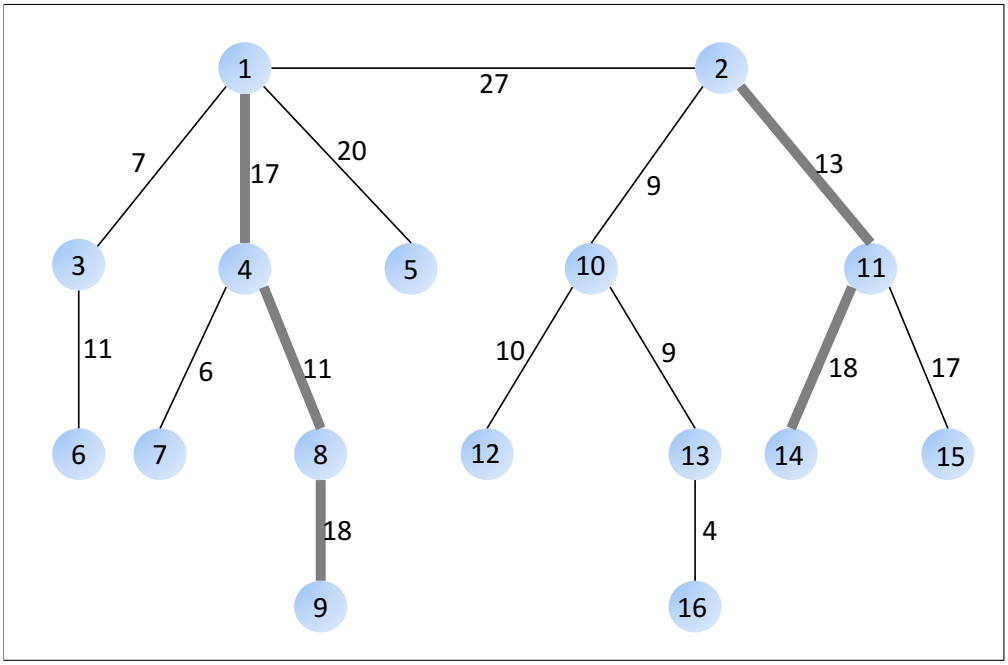


Figure 7.1: Communication Complexity

be varied, to make the eroor computing dependent of these two, *sPCA* [22] reported the norm of the reconstruction error divided by the norm of the matrix made up of the randomly selected rows, which is:

$$e = ||Y_r - X_r * W^{-1}||_1 / ||Y_r||_1$$

Unfortunately the W here is being generated as a whole but in our case we will be working on a particular segment of W. Therefore, We will have in hand a particular segment of W namely W_i, a particular segment of Y namely Y_i and X as a whole. This will break the data consistency and our calculated e will be the desired one. Therefore, we were not able to compute original reconstruction error in this approach and examine the stop condition. Rather we used some approach used widely in Matlab. We checked the change in W newly generated with the previous one and found the point of convergence by having a change in W smaller than the tolerance. Initially we had two variables:

$$dw = 0$$
$$maxWNew = 0$$

At the end of each iteration, These two variables will have their changed value and as soon as dw is less than tolerance the algorithm will be considered to achieve convergence.

$$maxWnew = max(|\widetilde{W}_{pq}|, maxWnew)\ \forall p \in D, q \in d$$
$$dw = max(|W_{pq} - \widetilde{W}_{pq}|, dw)$$
$$dw = \frac{dw}{\sqrt{eps} + maxWnew}$$

Algorithm 8 checkStopCondition

Input:$start$, end, $Wold$, $Wnew$, $stopCondition$, dw, $maxWNew$, $tolerance$
for p = $start$ to end **do**
  for q = 0 to nPC **do**
    $maxWnew$ = ***max***(***abs***($Wold[p - start][q]$), $maxWnew$)
  end for
end for
for p = $start$ to end **do**
  for q = 0 to nPC **do**
    $maxWnew$ = ***max***(***abs***($Wold[p - start][q] - Wnew[p - start][q]$), dw)
  end for
end for
$sqrtEps$ = 2.2204e-16
dw = dw/ ($sqrtEps + maxWnew$)
if $dw \leq tolerance$ **then**
  $stopCondition$ = $True$
end if

Chapter 8

Implementation in Spark Cluster System

We implemented our proposed system on Spark [35] cluster computing system. Spark gives us various types of storage system. Spark provides resilient distributed datasets (RDDs) and parallel operations on these datasets. Resilient Distributed Datasets (RDD) is a fundamental data structure of Spark [36]. It is an immutable distributed collection of objects. Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster. From user level the persistanc of RDD can be controlled. It can be cached in memory or store on disk to be used later through IO operations. Moreover the user can control its partitioning process like partition by key.

In case of small data that can be kept in memory while computing, we did the in-memory computations by making the input matrix Y persistent in the memory of the machines of our spark cluster. Therefore we could perform faster distributed operations on our data. On the other hand, for clusters with small count of nodes i.e. small amount of memory, we have to make IO operations by keeping the data on disks and load a specific segment of it while needed. The Algorithms 9 and 10 are based on the Spark Programming.

Algorithm 9 SegmentedXJob($X, Y, X_m, W, start, end$)

1: Y_nX = Y.zip(X)
2: X = Y_nX.map{$(Y_nX)_i \Rightarrow$
3: Y_i = $(Y_nX)_i$.*arg0()*.*range*($start, end$)
4: X_i = $(Y_nX)_i$.*arg1()*
5: dotRes = $Y_i \times W$
6: X_i = X_i + dotRes - $(X_m)_i$
7: }

Algorithm 10 SegmentedXJob($YtX, XtX, X, Y, Y_m, i, s, e$)

1: Y_nX = Y.zip(X)
2: **if** ($i == 1$) **then**
3: $YtXSum = spark.accumultor(newMatrix(D, d))$
4: $XtXSum = spark.accumultor(newMatrix(D, d))$
5: Y_nX.map{$(Y_nX)_i \Rightarrow$
6: Y_i = $(Y_nX)_i$.*arg0()*.*range*(s, e)
7: X_i = $(Y_nX)_i$.*arg1()*
8: $(YtX)_i$ = $Y_i^T \times X_i$ - $Y_m^T \times X_i$
9: $(XtX)_i$ = $X_i^T \times X_i$
10: $YtXSum$.*add*($(YtX)_i$)
11: $XtXSum$.*add*($(XtX)_i$)
12: }
13: $YtX = YtXSum$.*value()*
14: $XtX = XtXSum$.*value()*
15: **else**
16: $YtXSum = spark.accumultor(newMatrix(D, d))$
17: Y_nX.map{$(Y_nX)_i \Rightarrow$
18: Y_i = $(Y_nX)_i$.*arg0()*.*range*(s, e)
19: X_i = $(Y_nX)_i$.*arg1()*
20: $(YtX)_i$ = $Y_i^T \times X_i$ - $Y_m^T \times X_i$
21: $YtXSum$.*add*($(YtX)_i$)
22: }
23: $YtX = YtXSum$.*value()*
24: **end if**

Chapter 9

Experimental Evaluation

In this chapter we will present our experimental result that we found while implementing our approach in *Spark* Cluster computing system.

9.1 Cluster Setup

We built *Spark* clusters in our *Undergraduate Thesis Lab*. To evaluate the performance of the method to handle *Tall and Wide Big Data*, we created clusters that was composed of three machines with one master node and two slave nodes. Each of the machines had $64GB$ of memory and quad core processor of $3.6GHz$. That makes our cluster with 12 cores of processor to work with. We installed *Apache Spark 2.0* with *Hadoop 2.7* for *HDFS*.

On the other hand, to test the accumulation process, we created 8 spark clusters each of which contained only one slave node. as we did not have that amount of machines in a single cluster, we used small dimensional data to test this part of the method.

9.2 Data Sets

We use two real datasets. These data sets contains data of different types and of different sizes. The dimensionalities are also different. The datasets are:

- **Amazon Product Ratings**
  This data set is from *Amazon* web sites that includes the user ratings on various products items. In the raw data, it contains three columns: 1) unique User ID, 2) unique Product ID, 3) Product Rating. To make it easily fit in our system we gave each of the User ID and Product ID a number. There was $21\ millions$ user id's and $9\ millions$ product id's.

That makes it $21M \times 9M$ sparse data matrix. In the matrix if cell (x, y) have a number z, that means user x has given a rating z to the product y. The raw data was of size $4.73GB$

- **Twitter Follower Relationship**
  This data set is of *Twitter* that includes the follower information of every user ID. The raw data has two columns: two User IDs. The first User ID is followed by the second one. The maximum user Id is 61578414. That makes it a data set where every data sample has a dimension of $61M$ while it has $61M$ rows that makes it a data set of $61M \times 61M$. a 1 in cell (x, y) indicates that user y follows user x while a 0 indicates the negatively. The raw data was of size $26.17GB$

9.3 Performance Metrics

We did not have much opportunity to compare our method with others as there are no other algorithm that can handle geo-distributed tall and wide big data. Though the *sPCA* [22] showed some kind of implementation of PPCA in a distributed cluster, it also fails to run on very large dimensional data. Moreover, it has no method to accumulate partial data from different geographic locations.

9.3.1 Partition Count of W

In case of handling Tall and Wide data in a single cluster we made partitions of the principal subspace W. However, one of the limitations of our method is that we could not find a optimum partition count based on the data size and available resources. Rather we gathered some empirical results by giving manual input on the partition count. Table 9.1 shows the influence of no of partitions on the total running time and IO operation time of the data matrix of *Amazon Product Rating* Dataset. As at a time, we are working on a single partition of W and also creating a single partition of W, we had to allow IO operations. We had to load and store W_i while working on the range of dimension of our data matrix Y corresponding to the subscript i. Table 9.2 shows the same information for data matrix of *Twitter Follower* Dataset.

9.3.2 Running Time

We compare our Spark implementation with *sPCA*. It should be noted that for higher dimensional data *sPCA* will fail to run. As a result, we will not be able to make a comparison for such data samples.

Memory	Data Size	Partitions	Iterations	Runtime	IO Time
$6GB$	$21M \times 9.8M$	15	7	18.0 Hrs	65 Mins
		13	8	18.5 Hrs	50 Mins
		11	7	14.1 Hrs	44 Mins
		10	Failed		
$8GB$	$21M \times 9.8M$	12	8	9.8 Hrs	61 Mins
		11	7	9.5 Hrs	57 Mins
		10	6	5.5 Hrs	50 Mins
		9	Failed		
$8GB$	$10M \times 7M$	9	8	5.3 Hrs	41 Mins
		8	7	4.1 Hrs	38 Mins
		7	Failed		

Table 9.1: Effect of Partition count of W on Amazon Data Set

Memory	Data Size	Partitions	Iterations	Runtime	IO Time
$8GB$	$61M \times 61M$	22	6	91.1 Hrs	345 Mins
		20	7	108.5 Hrs	397 Mins
		19	Failed		
$12GB$	$61M \times 61M$	20	8	99.3 Hrs	405 Mins
		18	7	87.4 Hrs	317 Mins
		17	Failed		

Table 9.2: Effect of Partition count of W on Twitter Data Set

Time to Achieve Target Accuracy

We compare our spark implementation with sPCA-Spark based on the time needed to reach our desired convergence point. We varied the size of the input dataset in two ways 1) change dimension size for fixed sample count and 2) change sample count for a fixed dimesnion size. Then we measured the time needed for our implementaion as well as *sPCA-Spark*. The comparison results is shown in Figures 9.1 and 9.2. We used the dataset of **Amazon Product Ratings**.

In Figure 9.1 we varied the sample count of the input data matrix. However, we kept the same dimension i.e. column count (80 thousands in number). The figure shows that the running times for our implementation is higher than that of the *sPCA*. We had a different approach of implementation that increases the sequential tasks in our algorithm. We had to make this trade off in order to handle the higher dimension of data.

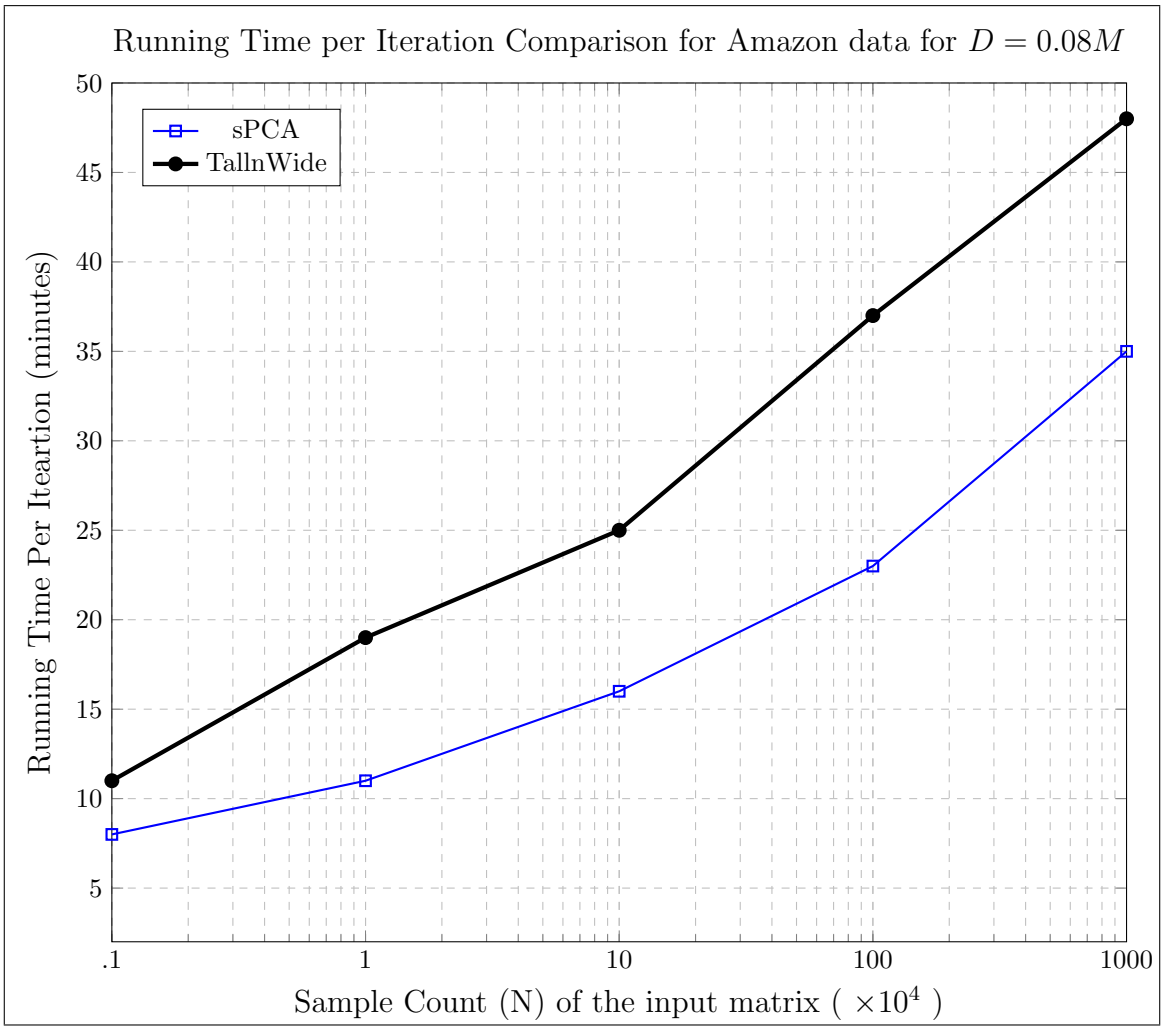


Figure 9.1: Running Time per Iteration Comparison for Amazon data for $D = 0.08M$

In Figure 9.2 we varied the dimension of each sample of the input data matrix. However, we kept the sample count i.e. row count (21 millions in number). The figure shows that the running times for our implementation is little bit higher than that of *sPCA*. However *sPCA* failed to run

for data with dimension larger than $140K$ and our algorithm was successful to perform PPCA on data with those higher dimension showed in the graph. The graph shows dimension up to $2M$ while for this dataset we run our algorithm for data with dimension up to $9.8M$.

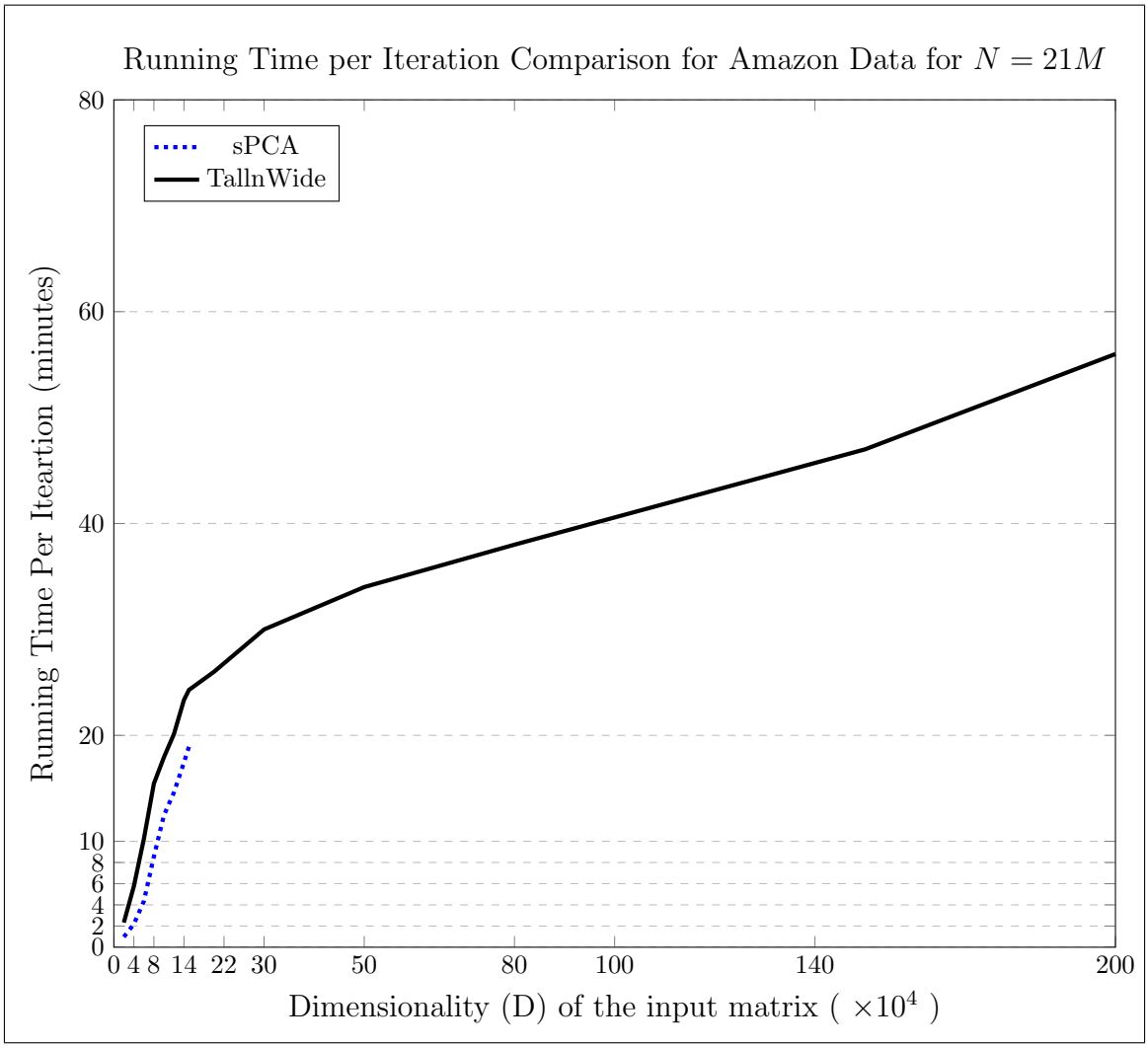


Figure 9.2: Running Time per Iteration Comparison for Amazon Data for $N = 21M$

Same two comparison on Twitter daata set are shown in Figure 9.3 and 9.4

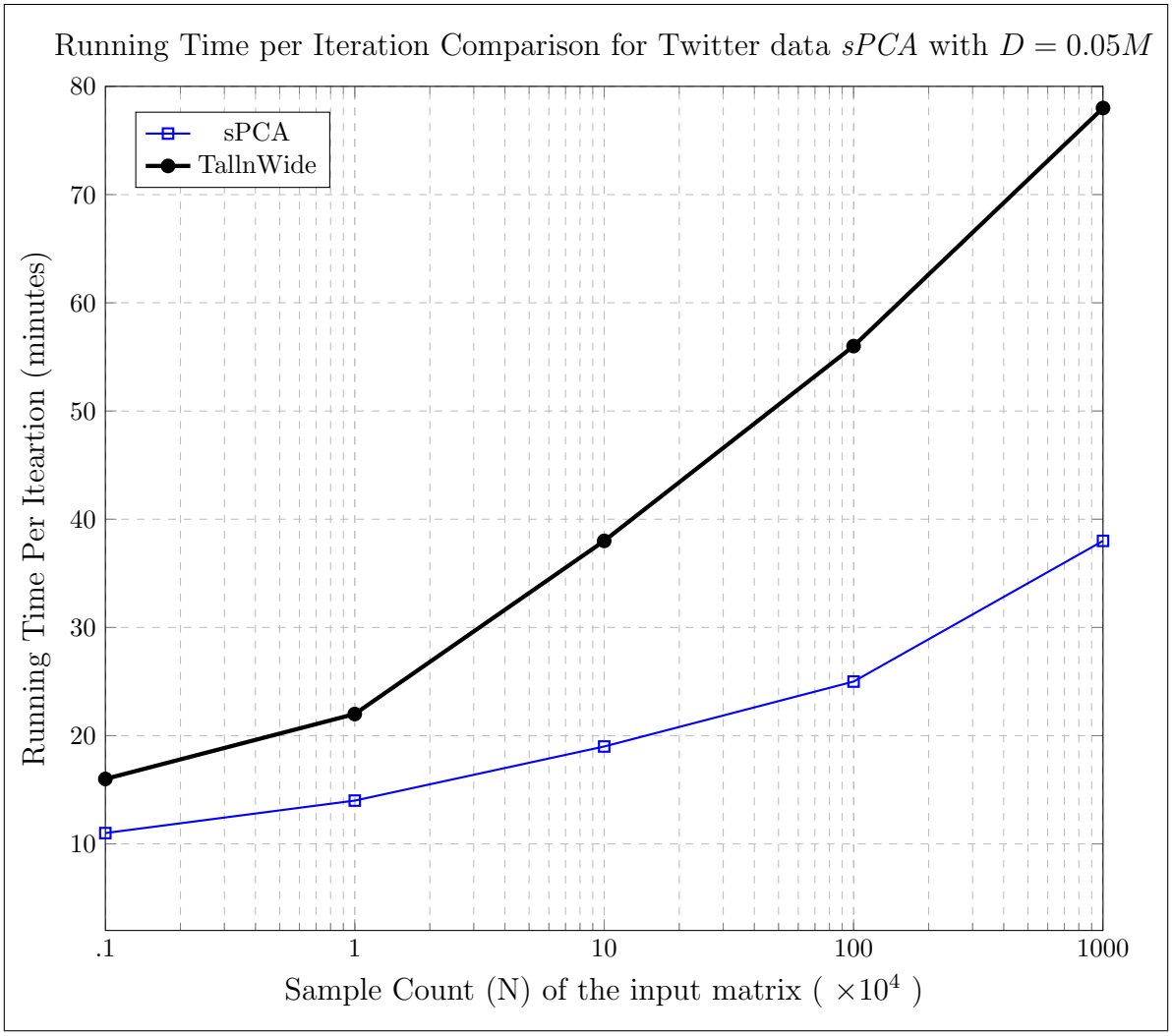


Figure 9.3: Running Time per Iteration Comparison for Twitter data *sPCA* with $D = 0.05M$

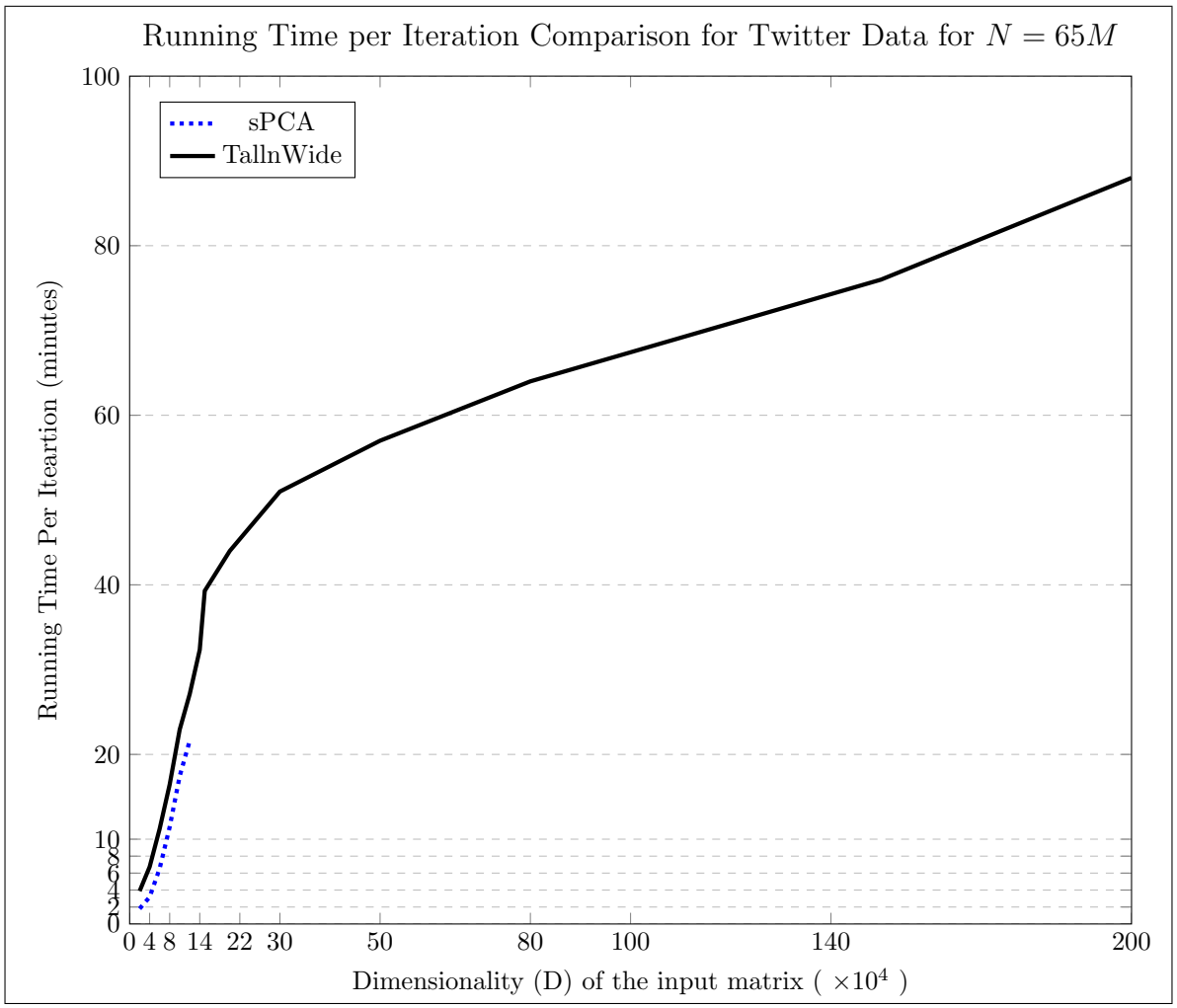


Figure 9.4: Running Time per Iteration Comparison for Twitter Data for $N = 65M$

Memory Usage

Figure 9.5 shows the memory utilization during the generation of a single segment of W on Amazon Data Set. The graph is shown for $N = 21M$. The algorithm was run on $8GB$ memory. In the graph any segment count on the left of the left most point on each curve willl result in failure of execution. It also shows that memory utilization becomes saturate at minimum of around $2GB$ as the segment count converges to 17 in this setup.

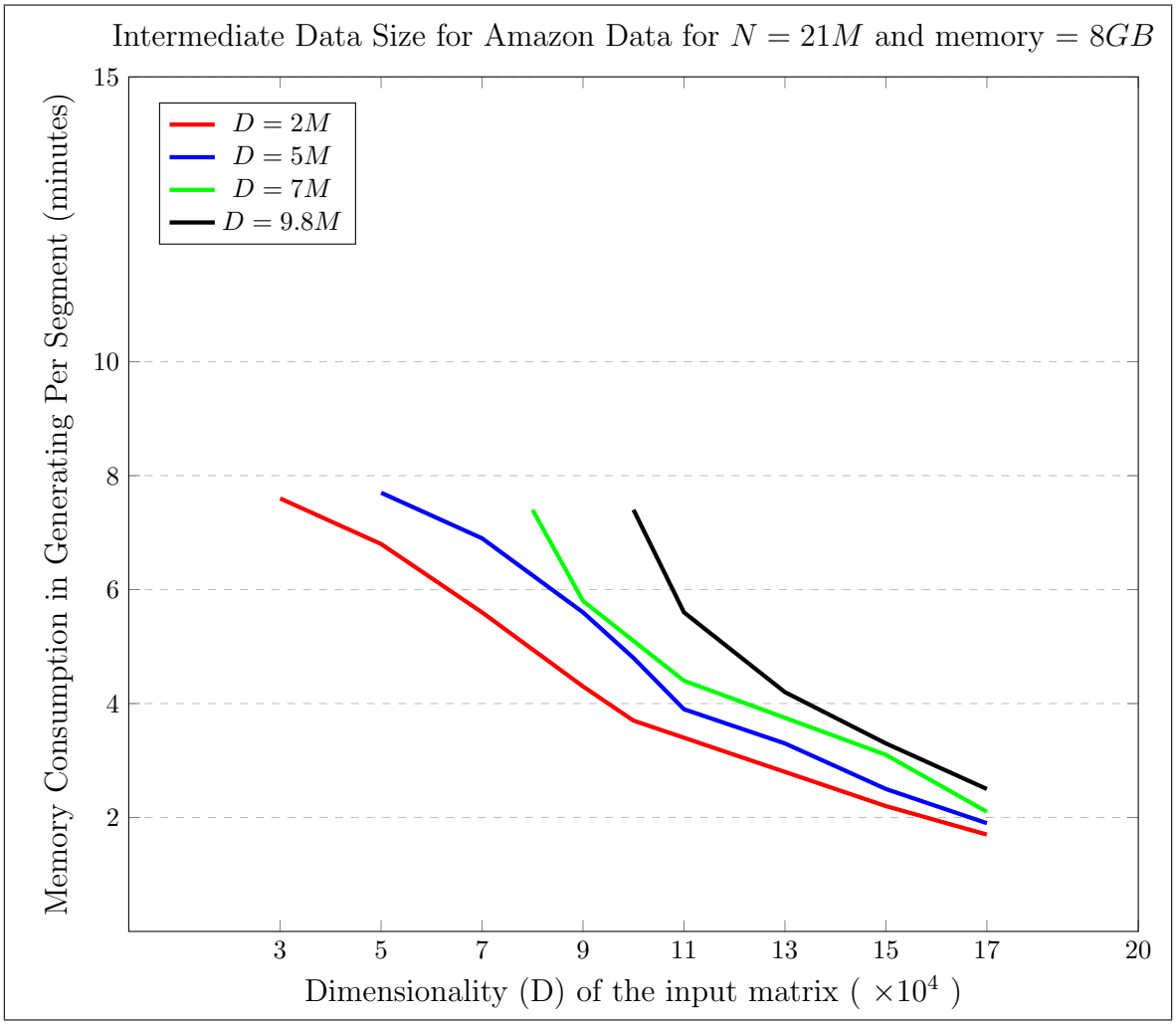


Figure 9.5: Intermediate Data Size for Amazon Data for $N = 21M$ and memory = $8GB$

Chapter 10

Conclusion

In this chapter we draw our concluding remarks and some future works that can possibly improve our present work at a large scale.

10.1 Concluding remarks

In this paper, we tried to analyze various issues of computing the principal components of an input data matrix which has large number of data samples while each of the samples has very large dimension. We characterized this kind of data as *Tall and Wide Big Data*. In addition the data is distributed at different geographic locations. Our analysis also indicated that the present algorithms for performing *PCA* are not capable of performing the task on such data sets. We presented a new approach and implementation for PCA that is capable of handling big data with very large dimension and can accumulate the partial results from different geographic locations in an efficient way. Our approach is based on the probabilistic PCA (PPCA) algorithm defined in [32]. We implemented this method on Spark platform and showed that it did the job significantly well.

10.2 Future Works

There are some issues that we were not capable of solving. Therefore, in future these issues can be handled to make our method a more effective one.

10.2.1 Capability of Computing Accuracy and Error

As stated earlier we could not calculate the error and accuracy. In [22], they measure the accuracy by computing the 1-Norm of the reconstruction error, which is given by: $e = ||Y - X * W^{-1}||_1$. Although this provides a common way to compare the accuracy of different algorithms, the reconstruction error is a big, dense matrix which is costly to store and process. They reduce the cost of storage by computing the error row by row, avoiding the need to store the large reconstruction matrix in the file system. Nevertheless, iterating over the resulting dense rows is still time consuming. They reduce this time by measuring the error only on a random subset of the rows. To have a unique way to interpret the measured error, independent of the sampling rate or matrix size, they report the norm of the reconstruction error divided by the norm of the matrix made up of the randomly selected rows, which is:

$$e = ||Y_r - X_r * W^{-1}||_1 / ||Y_r||_1$$

In addition, they measure the ideal accuracy that can be achieved with 50 principal components after a large number of iterations. After each iteration, they report the percentage of the ideal accuracy that is achieved.

However, in our case as we are working with very large sized data and so even after applying all the improvements, we failed to find the actual error and accuracy. Therefore, we had to use different approach to check the convergence.

10.2.2 Designing Better Stop Condition

The stop condition was not that much up to the mark in our method. As we were not able to find the actual accuracy and error rate, it results in an alternative approach in stop condition. However, using current accuracy and actual accuracy ratio a better stop condition can be designed.

10.2.3 Designing Optimum Partition Count of Principal Subspace W

In case of handling Tall and Wide data in a single cluster we made partitions of the principal subspace W. However, one of the limitations of our method is that we could not find a optimum partition count based on the data size and available resources. Rather we gathered some empirical results by giving manual input on the partition count. In future it would be a good work to find a formula to calculate the minimum number of partition at which the cluster can handle that particular wide data.

10.2.4 Improved Implementation for Dense Matrix Data

We did our evaluation on sparse matrix. However, we did not give proper and optimized implementation for dense matrices. In future we intend to do the optimization so that it can handle tall and wide dense data also.

References

[1] Wikipedia link: https://en.wikipedia.org/wiki/Data_analysis

[2] "Analyze quality in life in u.s. cities using pca - matlab and simulink".

[3] http://whatis.techtarget.com/definition/data-sovereignty

[4] https://en.wikipedia.org/wiki/Data_parallelism

[5] https://en.wikipedia.org/wiki/Normal_distribution

[6] https://www.quora.com/What-is-the-difference-between-model/-parallelism-and-data-parallelism

[7] https://www.tensorflow.org/

[8] Matei Zaharia et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In NSDI, 2012.

[9] Yucheng Low et al. Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud. PVLDB, 2012.

[10] Mu Li et al. Scaling distributed machine learning with the parameter server. In OSDI, 2014.

[11] Ashish Thusoo et al. Data warehousing and analytics infrastructure at Facebook. SIGMOD, 2010.

[12] George Lee et al. The unified logging infrastructure for data analytics at Twitter. PVLDB, 2012.

[13] Aditya Auradkar et al. Data infrastructure at linkedIn. In ICDE, 2012.

[14] Ariel Rabkin et al. Aggregation and degradation in jetstream: Streaming analytics in the wide area. In NSDI, 2014.

[15] Ashish Vulimiri et al. Global analytics in the face of bandwidth and regulatory constraints. In NSDI, 2015.

[16] Nikolaos Laoutaris et al. Inter-datacenter bulk transfers with netstitcher. In SIGCOMM, 2011.

[17] Albert Greenberg et al. The cost of a cloud: research problems in data center networks. SIGCOMM, 2008.

[18] Martin Rost and Kirsten Bock. Privacy by design and the new protection goals. DuD, January, 2011.

[19] European Commission press release. Commission to pursue role as honest broker in future global negotiations on internet governance. http://europa.eu/rapid/press-releaseIP-14-142en.htm.

[20] Principal component analysis: https://en.wikipedia.org/wiki/Principal_component_analysis.

[21] J. Shlens. A tutorial on principal component analysis. *arXiv preprint arXiv:1404.1100*. 2014.

[22] T. Elgamal and M. Hefeeda. Analysis of pca algorithms in distributed environments. *arXiv preprint arXiv:1503.05214*. 2015.

[23] N. P. Halko. Randomized methods for computing low-rank approximations of matrices. Ph.D. dissertation, Boulder, CO, USA, 2012, aAI3507998.

[24] M. E. Tipping and C. M. Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology), vol. 61, no. 3, pp. 611?622*. 1999.

[25] S. Roweis. Em algorithms for pca and spca. *Advances in neural information processing systems, pp. 626?632* 1998.

[26] I. Jolliffe. Principal component analysis. 1986. 1986.

[27] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan. MLbase: A distributed machine-learning system. In *Proc. Conf. on Innovative Data Systems Research (CIDR), 2013*.

[28] G. Golub and C. E. Reinsch. Singular value decomposition and least squares solutions. *Numerische Mathematik*, 14(5), 1970.

[29] J. Demmel and W. Kahan. Accurate singular values of bidiagonal matrices. *SIAM J. Sci. Stat. Comput*, 11(5), 1990.

[30] V. Hernandez, J. Roman, and A. Tomas. A robust and efficient parallel SVD solver based on restarted Lanczos bidiagonalization. *Electronic Transactions on Numerical Analysis*, 31, 2008.

[31] N. P. Halko. Randomized methods for computing low-rank approximations of matrices. *PhD thesis, University of Colorado*, 2012.

[32] M. E. Tipping and C. M. Bishop. *Mixtures of probabilistic principal component analysers. Neural Computation, 11(2)*, 1999.

[33] Nokia White Paper. *Datacenter interconnect market trends and requirements*. weblink: https://resources.ext.nokia.com/?cid=181666

[34] Fujitsu White Paper. *Application Note: Data Center Interconnect*. weblink: http://www.fujitsu.com/us/Images/Data-Center-Interconnect-app-note.pdf

[35] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In Proc. *USENIX Conf. on Networked Systems Design and Implementation (NSDI)*, NSDI12. USENIX Association, 2012.

[36] Tutorials Point Tutorial. weblink: https://www.tutorialspoint.com/apache_spark/apache_spark_rdd.htm

[37] Tutorials Point Tutorial. weblink: https://www.tutorialspoint.com/hadoop/hadoop_mapreduce.htm

[38] Soumen Chakrabarti, James Demmel, Katherine Yelick. Modeling the benefits of mixed data and task parallelism. *SPAA '95 Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures. Pages 74-83*

Appendix A

Linear Algebra

This section proves a few unapparent theorems in linear algebra, which are crucial to this thesis.

A.1 The inverse of an orthogonal matrix is its transpose.

Let A be an $m \times n$ orthogonal matrix where a_i is the i^{th} column vector. The ij^{th} element of $A^T A$ is

$$(A^T A)_{ij} = a_i^T a_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

Therefore, because $A^T A = I$, it follows that $A^{-1} = A^T$.

A.2 For any matrix A, $A^T A$ and AA^T are symmetric.

$$\boldsymbol{(AA^T)^T = A^{TT}A^T = AA^T}$$

$$\boldsymbol{(A^T A)^T = A^T A^{TT} = A^T A}$$

A.3 A matrix is symmetric if and only if it is orthogonally diagonalizable.

Because this statement is bi-directional, it requires a two-part "if-and-only-if" proof. One needs to prove the forward and the backwards "if-then" case.

Let us start with the forward case. If A is orthogonally diagonalizable, then A is a symmetric matrix. By hypothesis, orthogonally diagonalizable means that there exists some E such that $A = EDE^T$, where D is a diagonal matrix and E is some special matrix which diagonalizes A. Let us compute A^T.

$$\boldsymbol{A^T = (EDE^T)^T = E^{TT}D^TE^T = EDE^T = A}$$

Evidently, if A is orthogonally diagonalizable, if must also be symmetric.

The reverse case is more involved and less clean so it will be left to the reader. In lieu of this, hopefully the "forward" case is suggestive if not somewhat convincing.

A.4 A symmetric matrix is diagonalized by a matrix of its orthonormal eigenvectors.

Let A be a square $n \times n$ symmetric matrix with associated eigenvectors $\{e_1, e_2, ..., e_n\}$. Let $E = [e_1 e_2 ... e_n]$ where the i^{th} column of E is the eigenvector e_i. This theorem asserts that there exists a diagonal matrix D such that $A = EDE^T$.

This proof is in two parts. In the first part, we see that the any matrix has the special property that all of its eigenvectors are not just linearly independent but also orthogonal, thus completing our proof.

In the first part of the proof, let A be just some matrix, not necessarily symmetric, and let it have independent eigenvectors (i.e. no degeneracy). Furthermore, let E = [e1 e2 ::: en] be the matrix of eigenvectors placed in the columns. Let D be a diagonal matrix where the ith eigenvalue is placed in the iith position.

We will now show that AE = ED. We can examine the columns of the right-hand and left-hand sides of the equation.

Let hand side: $AE = [Ae_1 Ae_2 ... Ae_n]$

Right hand side: $ED = [\lambda_1 e_1 \lambda_2 e_2 ... \lambda_n e_n]$

Evidently, if $AE = ED$ then $Ae_i = \lambda_i e_i$ for all i. This equation is the definition of the eigenvalue equation. Therefore,it must be that $AE = ED$. A little rearrangement provides $A = EDE^{-1}$, completing the first part the proof.

For the second part of the proof, we show that a symmetric matrix always has orthogonal eigenvectors. For some symmetric matrix, let λ_1 and λ_2 be distinct eigenvalues for eigenvectors e_1 and e_2.

$$\begin{aligned}\lambda_1 e_1.e_2 &= (\lambda_1 e_1)^T e_2 \\ &= (Ae_1)^T e_2 \\ &= e_1^T A^T e_2 \\ &= e_1^T A e_2 \\ &= e_1^T (\lambda_2 e_2) \\ \lambda_1 e_1.e_2 &= \lambda_2 e_1.e_2\end{aligned}$$

By the last relation we can equate that $(\lambda_1 - \lambda_2)e_1.e_2 = 0$. Since we have conjectured that the eigenvalues are in fact unique, it must be the case that $e_1 e_2 = 0$. Therefore, the eigenvectors of a symmetric matrix are orthogonal.

Let us back up now to our original postulate that A is a symmetric matrix. By the second part of the proof, we know that the eigenvectors of A are all orthonormal (we choose the eigenvectors to be normalized). This means that E is an orthogonal matrix so by theorem 1, $E^T = E^{-1}$ and we can rewrite the final result.

$$A = EDE^T$$

Thus, a symmetric matrix is diagonalized by a matrix of its eigenvectors.

Generated using Undegraduate Thesis LaTeX Template, Version 1.3. Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh.

This thesis was generated on Sunday 10th September, 2017 at 8:30pm.