A GENERAL SUMMARIZATION MATRIX FOR SCALABLE MACHINE LEARNING MODEL COMPUTATION IN THE R LANGUAGE

A Thesis Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Siva Uday Sampreeth Chebolu

May 2019

A GENERAL SUMMARIZATION MATRIX FOR SCALABLE MACHINE LEARNING MODEL COMPUTATION IN THE R LANGUAGE

Siva Uday Sampreeth Chebolu

APPROVED:

Dr. Carlos Ordonez, Chairman
Dept. of Computer Science

Dr. Christoph F. Eick
Dept. of Computer Science

Dr. Klaus Kaiser
Dept. of Mathematics

Dean, College of Natural Sciences and Mathematics

Acknowledgements

I received a great deal of support and assistance with all of the writing and dissertation tasks. I would like to first thank my advisor Dr. Carlos Ordonez for his invaluable expertise, especially in formulating a research topic and methodology. He is one of the smartest people I have met. He also provided very insightful discussions, not only about my research, but also about new technologies and trends that have recently evolved. Collectively, these discussions improved my confidence to successfully complete the research.

I am also very grateful to Dr. Christopf F. Eick and Dr. Klaus Kaiser for taking time to be part of my thesis committee and for sharing their knowledge and experience on my research.

It has been a great time working with my fellow researchers: Tahsin, Steve, and Xiantian. I also wish to thank my dear friends, Vishal and Ravali, who helped me to smoothly complete my research.

A GENERAL SUMMARIZATION MATRIX FOR SCALABLE MACHINE LEARNING MODEL COMPUTATION IN THE R LANGUAGE

An Abstract of a Thesis

Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Siva Uday Sampreeth Chebolu

May 2019

Abstract

Data analysis is an essential task for research. Modern large datasets indeed contain a high volume of data and may require a parallel DBMS, Hadoop Stack, or parallel clusters to analyze them. We propose an alternative approach to these methods by using a lightweight language/system like R to compute Machine Learning models on such datasets. This approach eliminates the need to use cluster/parallel systems in most cases, thus, it paves the way for an average user to effectively utilize its functionality. Specifically, we aim to eliminate the physical memory, time, and speed limitations, that are currently present within packages in R when working with a single machine. R is a powerful language, and it is very popular for its data analysis. However, R is significantly slow and does not allow flexible modifications, and the process of making it faster and more efficient is cumbersome. To address the drawbacks mentioned thus far, we implemented our approach in two phases. The first phase dealt with the construction of a summarization matrix, Γ, from a one-time scan of the source dataset, and it is implemented in C++ using the RCpp package. There are two forms of this Γ matrix, Diagonal and Non-Diagonal Gamma, each of which is efficient for computing specific models. The second phase used the constructed Γ Matrix to compute Machine Learning models like PCA, Linear Regression, Naïve Bayes, K-means, and similar models for analysis, which is then implemented in R. We bundled our whole approach into a R package, titled Gamma.

Contents

List of Tables

Chapter 1

Introduction

1.1 Problem Importance

Machine Learning has become popular and has gained widespread demand due to the availability of abundant data and processing power. Many tools and technologies, like Python, R, Scala, Java, and C#, compute these Machine Learning models. Datasets can be so large that they do not fit in the tools' main memory. For these types of data, Hadoop Stack, distributed systems, or columnar databases like Vertica are popular choices to compute Machine Learning models. We propose that the size of the cleaned dataset, rather than its raw counterpart, should dictate which data processing platform is to be used [1]. Data cleaning strips unwanted and inaccurate data. As a result, the size of the dataset is significantly reduced, along with the need to use a heavyweight data processing platform like Hadoop. Therefore, with a refined dataset, data processing can be limited to a single-system environment like

R.

1.2 Motivation

The average user, although they may have a large dataset that does not fit in their tool's main memory, may not use heavyweight data processing systems like Hadoop because of the installation process, maintenance time, and cost. The downside of using lightweight data processing systems like R or Python is that they require all the data to be stored in the memory to compute Machine Learning models, which is practically impossible for the average user. Our package, to a great extent, solves these issues by giving support to computing four fundamental Machine Learning models Linear Regression, Principal Component Analysis (PCA), Naïve Bayes, and K-Means without main memory limitations. These models are considered to be classical Machine Learning models that fall into the categories of Regression Algorithms, Dimensionality Reduction Algorithms, Bayesian Algorithms, and Clustering Algorithms, respectively.

1.3 Scope

R is a vast package ecosystem coupled with extensive developer support, and it ticks all the necessary boxes as a data analytics platform. Novel research methods in the field of Machine Learning likely have a readily available package in R. However, R has a few shortcomings in areas like memory management, speed, and efficiency.

While parallelism in R can be achieved using packages like *parallel*, the shortcomings become evident with an increasing number of cores. The language design sometimes poses a great problem when working with large datasets, since the data has to be stored in the physical memory. With the dedicated physical memory of a system, R cannot scale to work with datasets larger than the proportion of memory allocated to it, in these cases, it is forced to crash. The physical memory limitation clearly outweighs the need for parallelization in R, and this limitation is our major research focus. Our research is important for classical linear Machine Learning algorithms, rather than non-linear algorithms like Support Vector Machines (SVM), Deep Neural Networks, Logistic Regression, and Hidden Markov models. The mathematical part of the research proved to be challenging in contrast to its implementation. We assumed that the model from our package would comfortably fit in the main memory. Our model has a few benefits. First, the environment does not crash even when using large datasets, which is a downside of existing R packages. Second, it works independent of the physical memory allocated to the R environment, thus eliminating the physical memory limitations. Third, it produces as accurate results as the existing packages that compute the above models in R. Finally, our package works faster than the current packages in R, which is demonstrated in Chapter 6.

Chapter 2

Related Work

This thesis is based on several previous research achievements. The Summarization Matrix described in this research was first proposed was first proposed by Ordonez and colleagues [2] and then deeply explored by the same group [3] to propose two forms of matrices based on the type of data, namely Dense and Sparse. Further, [3] explored the optimizations for Sparse input datasets and showed that their SciDB implementation defeated Spark performance on a large cluster. Then, this work was migrated to a columnar Database System in work by Zhang and colleagues [4].

2.1 Computing Scalable Machine Learning Models

Many advancements have recently been made in the computation of Machine Learning models. No matter how big the dataset size or how complex the determination of relationship among the attributes of the datasets, we are able to see through them. To turn Machine Learning algorithms into scalable ones, many researchers have developed a variety of novel techniques and different optimization mechanisms. A combination of different methods with frameworks or platforms were used in a work by Rehab and Boufares [5], which combines MapReduce, for Multiple Linear Regression based on the QR decomposition, and the ordinary least squares method. Recent trends aim to incorporate data analytical capabilities within Database Systems [6]; these provide a predictive analytical model, and the associated regression query processing algorithms reveal dependencies between the values of different attributes. This was implemented on databases, like SciDB, to compute different models like Linear Regression, PCA, and Variable selection, which are achieved with a single pass on the entire dataset [3]. Many improvements have been made on the existing algorithms, like K-Means, by determining the number of clusters automatically, thereby eliminating its specification when we start running the model. In [7], the authors achieved this improvement by developing an algorithm in MapReduce and improving the quality of clustering. Developing new techniques and optimizing existing algorithms contribute to scalable Machine Learning.

2.2 Gram Matrix in Machine Learning

In linear algebra, the Gram matrix, also called the Gramian matrix or Gramian, of a set of vectors $v_1 \dots v_n$ in an inner product space is the Hermitian matrix of inner products, whose entries are given by $G_{ij} = \langle v_i, v_j \rangle$ from [8]. For finite-dimensional real vectors with the usual Euclidean dot product, the Gram matrix is simply $G = V^T V$, where V is a matrix whose columns are the vectors v_k. This Gram matrix can be applied to compute the linear independence of vectors; that is, a set of vectors are linearly independent if and only if the Gram determinant (the determinant of the Gram matrix) is non-zero.

This Gram matrix can be used for many more applications. Because they use a well-conditioned Gram matrix, the Linear and Quadratic programming algorithms, when applied to the M-MIMO detection problem, provide interesting results, such as exploitation of the structure of the problem (simple constraints) and improvement of the rate of convergence, as mentioned by Fukuda and Abro [9]. This Gram matrix can also be applied for Observability Analysis. It can be used to verify local redundancy, which is important in measurement system planning. Determination of non-redundant pseudo-measurements for merging observable islands into an observable system is carried out by analyzing the pivots of the Gram matrix [10].

Our summarization approach contains this Gram matrix as one of its sub-matrices, Q, which is described in Section 4.3.

2.3 Comparing Similar Implementations and Systems

In the previous section, we discussed some techniques that are generally used to compute Scalable Machine Learning models. There are many techniques to improve the performance of the Linear Regression, PCA, Naïve Bayes, and K-Means models, a few of which are briefly stated in this section. Data is growing exponentially larger, may not fit in the model's main memory, and may require an external data storage platform to compute the Machine Learning models. Existing research has computed the models using efficient data summarization, similar to the research in this thesis. A framework from Lai and colleagues [11] is based on the notion of data space reduction, which finds high-density areas in a given feature space. The dense cells store summarized information of the data on which a designated partitioning or hierarchical clustering algorithm, like K-Means, can be used as the next step to find the clusters.

A Linear Regression Algorithm is used to achieve parallelism in database processing, while Compute Unified Device Architecture (CUDA) uses a multi-threading technique [12]. By utilizing this technique, Linear Regression is bound to achieve a very high performance when compared to data processing on other platforms. Both performance and accuracy matter when computing Machine Learning models. In a publication by Senavirathne and Torra [13], a linear regression approximation method was implemented based on integral privacy, which ensures high accuracy and

robustness while maintaining a degree of privacy for ML models. There are decent enhancements for the Naïve Bayes model like those presented by Vilalta and Rish [14], which used decomposition of classes via clustering to improve Naïve Bayes.

Many other works used triangle inequality, collaboration of compressed sensing theory, and the K-SVD approach to accelerate k-means [15, 16, 17]. If we observe carefully, Linear Regression, Naïve Bayes, or PCA do not require initialization, unlike the K-Means model, which requires the number of clusters and their respective centroids to be initialized. If initialization is bad, there is no convergence on a solution.

Scalable Machine Learning algorithms were summarized in parallel [3]. Ordonez and colleagues [3] exploited HP Vertica's parallelization feature, similar to works by Lin and Raychev [18, 19], to simultaneously perform summarization on multiple systems. We adapted the algorithms from the Ordonez publication [3] and implemented them such that they are serial, scalable, and 99 percent accurate in R. We made use of R's chunking ability to read an infinite amount of input data, which made the process faster. We completely removed the use of the database system, which was the main component used by Ordonez and colleagues [3]. Pitchaimalai and colleagues [20] computed Naïve Bayes inside the database with pure SQL queries. We adapted this model computation [20] and implemented it in R. We compared our work with the most efficient packages in R, and we found shown that our package is faster and more reliable than the former.

Improvements for K-Means can include developing an enhanced agglomerative fuzzy K-Means clustering method with MapReduce implementation on the Hadoop

platform as in Zhang and Wang's work [21] or developing algorithms to remove outliers from the dataset and automatically selecting the initial centroids, thereby stabilizing the result, as performed by Boukhdhir and colleagues [22]. Similarly, to improve the performance of data storing and indexing for Naïve Bayes in text classification, Zhou and colleagues [23] presented a new hash and a software implementation of Naïve Bayes classification. This implementation was mapped in the Topo-MapReduce model on a multicore processor with circuit switch and packet switching.

Notwithstanding the implementations in parallel environments, the research in this thesis is based on serial processing in an R environment. Since R is a functional language, a lot of effort is required to parallelize it so that the speedup is positive, rather than the existing negative speedup corresponding to an increased number of cores or systems.

2.4 Linear Models in R and Mathematical Systems for the Models

Optimization techniques, like the Gradient Descent Method used by Lee [24], give only the approximate values for some models, despite being fast. This optimization algorithm is used to find the coefficients of a function at cost. Gradient Descent is applied when the parameters cannot be calculated using linear algebra and must be searched for by an algorithm. A new paradigm of assigning weights for classification

was proposed by Lee [24]. In contrast to the other weighting methods, it was implemented on Naïve Bayesian learning in which optimal weights are calculated using a Gradient approach.

Another approach is a Laplacian method to compute or improve Machine Learning models. A Laplace Naïve Bayes model with mean shrinkage was proposed by Wu and colleagues [25]. This method uses a conditional distribution of the samples, as it is less sensitive to outliers compared to the general normal distribution.

A Dynamic Programming method can compute or build upon the Machine Learning models. A Machine Learning-based state-space approximate dynamic programming approach was proposed to solve the self-scheduling problem faced by power plants under an integrated energy and reserve market by Keerthisinghe and colleagues [26].

We can also use the Integer Programming method. Kacprzyk and Szkatula [27] proposed a new improved inductive learning method using the Integer Programming; it is used to derive classification rules that correctly describe the examples belonging to a class and do not describe examples that do not belong to that class. This problem is also included in the set of problems solved by the Genetic Algorithm.

Alternatively, Microsoft R Open is designed to include an updated R engine (R 3.2.2). It includes new fuzzy matching algorithms with the ability to write to databases via ODBC with a streamlined installation experience.

Chapter 3

Background

3.1 Definitions on the Models and Datasets

We use Θ to represent a statistical model in general. Θ can be a Linear Regression or PCA model, as well as any of the clustering and classification models, such as K-Means and Naïve Bayes. PCA is an unsupervised model to reduce dimensionality. Linear Regression is a fundamental supervised model, and its solution helps to understand and build other linear models. Naïve Bayes is another classic supervised model, and its solution assigns a numerical value between 0 and 1 to each class label to denote the probability of data belonging to a specific class. K-Means is a clustering algorithm, and its goal is to find groups within the data, with the number of groups represented by the K. The algorithm works iteratively to assign each data point to one of the K groups based on the features that are provided. Data points are clustered based on feature similarity.

We can use the same dataset for both Linear Regression and PCA. On the other hand, we use separate datasets for Naïve Bayes and K-Means, which are different from the datasets used for Linear Regression and PCA. The dataset requirements are mostly similar for Linear Regression and PCA, whereas they are different for Naïve Bayes and K-Means.

3.2 General Hardware and Software Information

We solved the problem at hand using a single system and single core following the serial processing techniques. Generally, systems used in serial processing are configured with a Linux operating system with 1 TB to 5 TB of disk space and 8 GB to 16 GB of physical memory. Recent systems consist of a minimum of 4 cores and may extend up to 64 cores. However, R does not scale its processing for an increased number of cores, which is explained in Section 3.3.

3.3 General Definitions of R System

In this section, we focus on the how data is processed, and we explain important details about R and its run-time that led us to choose it as our data processing platform.

In R, vectors are stored as one contiguous block allocated dynamically. Matrices are two dimensional arrays of real numbers, which are stored as one block in column major order and are also dynamically allocated. Lists are the most general ones of

the data structures and can have elements of diverse data types, including atomic data types and nested data structures. Finally, data frames are a list of columns of diverse data types, in which each column is a C array dynamically allocated. R is fundamentally functional, but it also incorporates imperative programming statements. R also incorporates object-oriented features that enable the creation of new data types, libraries, and reusable functions. R uses a dynamic interpreter, and it utilizes the C language for matrix and data frame operations and the LAPACK library for linear algebra and numerical methods. When R functions are called, the R run-time creates nested variable environments, which are dynamically scoped. R runs only on a single CPU thread, despite installing it on machines with numerous cores. Thus, it cannot be easily parallelized using the current package support.

Chapter 4

Summarization with a Matrix

4.1 Summarization Terms

Firstly, we define the inputs given to the models. The most obvious one is the input dataset, interpreted as a matrix, which is defined as a set of n *column* vectors. All of the models take a $d \times n$ matrix as input. Let the input dataset be defined as X, which is considered to have n points, and each point is a vector in R. Therefore, we view X as a wide rectangular matrix. In the case of Linear Regression and Principal Component Analysis, we take an extra dimension (for output variable Y) resulting a change in the dimensions of X to $(d+1) \times n$, which we call $\mathbf{X}$. We use *i=1....n* and *j=1.....d* as matrix subscripts. We augment an extra row of n 1s to X and call that as $\mathbf{Z}$ for mathematical convenience. The convention of using *column* vectors and *column*-oriented matrices is also for mathematical convenience. We assume that $d << n$, that is, the number of attributes in the dataset are of tens and hundreds at

most but not thousands and more. We perform the following matrix multiplication in a most efficient manner rather than its raw counterpart: $\mathbf{Z}.\mathbf{Z^T}$. By doing so, we reduce the whole problem to a mere matrix multiplication which returns us some parameters upon which we build to compute the models. We will delve deeper into the parameters and the results in the Section 4.3.

4.2 Main Memory Algorithms (Unoptimized)

There are many ways to compute the aforementioned models in R. The four most common and efficient methods that are used to compute each model are as follows.

4.2.1 The lm() function - Linear Regression

The most common, simple and moderately efficient method to compute the Linear Regression in R is the $lm()$ method. This method takes a few parameters as input and gives a simple regression as output. Two of the most used parameters are discussed below:

1. $formula$: this is the description of the model you want to build. Generally, it is of the format Y_{var} X_{var}, where Y_{var} is the dependent or the predicted variable and X_{var} is the independent or the predictor variable.

2. *data*: this is the variable which contains the input data in it.

4.2.2 The svd() function - Principal Component Analysis

We did not compute the PCA completely. Instead, we computed up to the Singular Value Decomposition for a given input dataset. To get that result, first we compute the correlation matrix for the given dataset using the *cor*(). This function takes the following parameters as input:

1. x: numeric matrix or a data frame.
2. *method*: indicates the correlation coefficient to be computed. The default is pearson correlation coefficient which measures the linear dependence between two variables. kendall and spearman correlation methods are non-parametric rank-based correlation test.

Then, we computed the Singular Value Decomposition on the correlation matrix generated from *cor*(). We used the $SVD()$ to compute the above, which have the following parameters:

1. x: a numeric or complex matrix whose SVD decomposition is to be computed.
2. *nu*: the number of left singular vectors to be computed.
3. *nv*: the number of right singular vectors to be computed.

4.2.3 The naivebayes() function - Naïve Bayes

The most recent usage of the Naïve Bayes algorithm is the *naive_bayes*() function. This function takes several inputs, of which most used are mentioned below, fits the Naïve Bayes model in which the predictor variables are assumed as independent within each class label and gives model as output.

1. x: matrix or dataframe with categorical or numeric predictors.
2. y: class vector.
3. $formula$: an object of class "formula" (or one that can be coerced to "formula") of the form class predictors.
4. *data*: matrix or dataframe with categorical numeric predictors.

4.2.4 The kmeans() function - K-Means

To compute the K-Means model in R, we use the *kmeans*() function which take the following inputs, performs clustering and returns the model as output.

1. x: numeric matrix of data, or an object that can be coerced to such a matrix (such as a numeric vector or a data frame with all numeric columns).
2. *centers*: either the number of clusters, say k, or a set of initial (distinct) cluster centres. If a number, a random set of (distinct) rows in x is chosen as the initial centres.

3. *iter.max*: the maximum number of iterations allowed.

4.3 Our Contribution/ Improved Algorithms

From the motivation discussed in Section 4.1, we introduce the two forms of Summarization Matrices that we explored, namely the Non-Diagonal Gamma Matrix and the Diagonal Gamma Matrix. The Non-Diagonal Gamma Matrix is nothing but the one named as Γ by by Ordonez and colleagues [28]. Then, we move on to presenting the other form of our Summarization Matrix , which we call the Diagonal Gamma Matrix. We asigned these aforementioned names to the matrices in our research based on their usage and mathematical form.

4.3.1 Non-Diagonal Gamma Matrix

4.3.1.1 Theory and Representation

First, we quickly review the statistics that are integrated to form the Non-Diagonal Gamma Matrix, which are:

$$n = |X|$$

$$L = \sum_{i=1}^{n} x_i$$

$$Q = XX^T = \sum_{i=1}^{n} x_i \cdot x_i^T$$

in which, X is the dataset, n counts total number of points in the dataset, L is a linear sum of x_i and Q is a vector outer product where x_i is multiplied by itself, i.e.

Q is nothing but the quadratic sum of x_i,

As defined earlier in Section 4.1, for Linear Regression and PCA, X is $d \times n$ and $\mathbf{Z}$ is $(d+2) \times n$. Matrix $\Gamma_{non-diag}$, which is defined below, is a fundamental Gamma function containing a complete, definite and sufficient summary of X to efficiently compute models like Linear Regression and PCA which were previously defined. We define another Gamma function, Diagonal Gamma, in Section 4.3.2.1 for models like Naïve Bayes and K-Means.

We review the two ways of representing $\Gamma_{non-diag}$ from the work by Ordonez and colleagues [3], which are: (1) vector-matrix and matrix-matrix multiplications form; (2) sums of vector outer products form.

$$\Gamma_{non-diag} = \begin{bmatrix} n & L^T & \mathbf{1}^T \cdot Y^T \\ L & Q & XY^T \\ Y \cdot \mathbf{1} & YX^T & YY^T \end{bmatrix} = \begin{bmatrix} n & \sum x_i^T & \sum y_i \\ \sum x_i & \sum x_i x_i^T & \sum x_i y_i \\ \sum y_i & \sum y_i x_i^T & \sum y_i^2 \end{bmatrix} \tag{4.1}$$

From [3], we can compute $\Gamma_{non-diag}$ in a single matrix multiplication utilizing $\mathbf{Z}$ like $\Gamma_{non-diag} = \mathbf{Z}\mathbf{Z}^T = \sum_{i=1}^{n} z_i \cdot z_i^T$. That is, the square of matrix $\mathbf{Z}$ gives us $\Gamma_{non-diag}$. $\Gamma_{non-diag}$ is significantly smaller when compared to X, so that it comfortably fits in main memory.

4.3.1.2 Models Based on Non-Diagonal Gamma

4.3.1.2.1 Linear Regression Let $X = \{x_1, \ldots, x_n\}$ be n points in the data, representing the rows of a csv file, in our case. Each of the n points have d explanatory variables. Let $Y = \{y_1, \ldots, y_n\}$ be a vector in a manner that each x_i is associated with y_i. Now, Linear Regression can be defined as a model that establishes relation between the dependent variable y and the d explanatory variables. From [3], standard definition of Linear Regression is given as $Y = \beta^T \mathbf{X} + \epsilon$, where β is the column vector of regression coefficients and ϵ represents the Gaussian error. As we discussed earlier, $\mathbf{X}$ is nothing but X augmented with a row of n 1s. From [29], β is defined as $\hat{\beta} = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}Y^T$. From the above discussed Non-Diagonal Gamma, we can rewrite this equation as $\hat{\beta} = Q^{-1}(\mathbf{X}Y^T)$.

Therefore, the LR algorithm becomes:

1. Compute $\Gamma_{non-diag}$.
2. Solve $\hat{\beta}$ utilizing $\Gamma_{non-diag}$.

4.3.1.2.2 Principal Component Analysis: PCA is mainly implemented on a dataset to reduce noise and redundancy of dimensions. This implementation was conducted by expressing the dataset X on a new orthogonal basis, which is a linear combination of original dataset. PCA can be computed on the covariance matrix, V, or the correlation matrix, ρ, of the dataset used in Hastie and colleagues work [30]. We can compute PCA using any of the metrics mentioned above. This model require two parameters. First is U, which is a set of d orthogonal vectors, principal

components of the dataset, ordered in decreasing order by their variance. Second is the diagonal matrix D^2 which contains the squared eigen values. From [3], we can compute ρ, the correlation matrix, from the two parameters, D and U as $\rho = UD^2U^T = (UD^2U^T)^T$.

Then we compute PCA by using Eigen decomposition of the ρ, which is a symmetric matrix factorization. In general, only k dimensions out of d are considered as principal components which carry the most important information about the variance of dataset. That is the remaining $d - k$ dimensions can be discarded safely to reduce d. The actual reduction of d to some lower dimensionality k (i.e. the k principal components) requires an additional matrix multiplication, which is simpler and faster than computing Θ. That is, we compute PCA from the correlation matrix by solving Singular Value Decomposition on it. For our convenience, we will address Singular Value Decomposition as SVD from hereon. Also, we express ρ in terms of the sufficient statistics extracted from $\Gamma_{non-diag}$, which can be used in computing the SVD, as: $\rho_{ab} = \frac{(nQ_{ab} - L_a L_b)}{(\sqrt{nQ_{aa} - L_a^2}\sqrt{nQ_{bb} - L_b^2})}$. The two phases in PCA are:

1. Compute Non-Diagonal Gamma.

2. Compute ρ (or covariance matrix V) and solve SVD of ρ (or V).

4.3.2 Diagonal Gamma Matrix

4.3.2.1 Theory and Representation

From Ordonez and colleagues [3], it is clear that even though Non-Diagonal Gamma functions are iterative algorithms, they avoid reading the entirety of a dataset at every iteration. However, that approach cannot be applied on models like Naïve Bayes or K-Means which require more than one summarization matrix and may also require reading the entire dataset more than once. Naïve Bayes requires g summarization matrices for a given dataset, where g is the number of unique class labels in the dataset. K-Means requires k matrices for summarization of a dataset with k as the number of clusters given by the user (i.e. one for each cluster). Furthermore, these models do not require complete computation of the Non-Diagonal Gamma as described in Section 4.3.1. This difference is because the LR and PCA are computed in rotated space whereas in NB and KM, we assume that the dimensions are independent, making Gamma diagonal. Therefore, we are required to build another function, Diagonal Gamma, which helps compute these models. We do not require the Y parameter for Naive Bayes and K-means as used in Linear Regression and PCA. The major difference between the two forms of Gamma is that we do not require parameters off the diagonal in Diagonal Gamma matrix as in the Non-Diagonal Gamma matrix. So, we need only a few parameters out of the whole Non-Diagonal Gamma, namely, n, L, L^T, Q. That is, we require only a few sub-matrices from Non-Diagonal Gamma, which can be visualized as:

$$\Gamma_{diag} = \begin{bmatrix} n & L^T & 0 \\ L & Q & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$Q = \begin{bmatrix} Q_{11} & 0 & 0..... & 0 \\ 0 & Q_{22} & 0..... & 0 \\ 0 & 0 & Q_{33}..... & 0 \\ 0 & 0 & 0..... & Q_{dd} \end{bmatrix}$$

Furthermore, from the Γ_{diag}, we can observe that if we compute the terms n, L, and Q we can get the whole matrix, as L^T can be obtained by copying the transpose of L. This is the major change in definition of the Non-Diagonal Gamma to that of the Diagonal Gamma. We developed a generalized solution for all the models mentioned in Section 4.3.1.2 and Section 4.3.2.2, which is presented below in the Algorithm 1.

4.3.2.2 Models Based on Diagonal Gamma

4.3.2.2.1 Naïve Bayes The input for this model is a dataset X and the output is a Naïve Bayes classification model which contains C(mean per dimension), R (variance per dimension), and W (prior per class).

First, we take the dataset X as input in chunks of fixed size. In each chunk, we split the data based on number of classes in the dataset. We compute one gamma

for each part of chunk and add these Γ matrices with respect to the classes to arrive at a final list of Γ matrices one for each class. We extract N_g, L_g, Q_g as defined in Section 4.3.1, from this final list of Γs. So, we arrive at lists of N_g, L_g, Q_g.

From these lists, we compute prior (π), mu (μ), and sigma (σ) per dimension of each item in the list separately like:

$$\pi_g = \frac{N_g}{n}$$

$$\mu_g = \frac{L_g}{N_g}$$

$$\sigma_g = \frac{Q_g}{N_g} - \frac{L_g L_g^T}{N_g^2}$$

where $N_g = |Xg|$ and we take diagonal of $L * L^T$ and Q, which can be treated as vectors instead of a matrix.

These are the 3 parameters included in the Naïve Bayes model. Then, we could predict class labels for new data using this model. For the prediction, for each point in the input data, we compute a probability value per class using the model parameters.

$$probability_{xi_{class}} = (1/\sqrt{2 * \pi * \sigma_{x_i}^2}) * e^{(-0.5*(x_i - \mu_{x_i})^2/\sigma_{x_i}^2)} \tag{4.2}$$

We assign that class with maximum probability from the computed probabilities to that respective point in the new data.

4.3.2.2.2 K-Means The input for this model is a dataset X and the number of clusters represented by k and the output is three matrices C, R, W, containing the means, the variances and the weights, respectively, for each cluster and partitions of

X into k subsets. As we discussed earlier, X represents all points in the dataset, N is the total number of points, L is the linear sum of points and Q is the squared sum of points in the dataset X. Using definitions from Section 4.3.1, we now define similar terms X_j, N_j, L_j, Q_j as the subset of X which belong to a single cluster, the total number of points per cluster ($|X_j|$), the sum of points in a cluster ($\sum_{\forall x_i \in X_j} x_i$) and the sum of squared points in each cluster ($\sum_{\forall x_i \in X_j} x_i x_i^t$) respectively. With these statistics, we compute C_j, R_j, W_j as:

$$C_j = \frac{L_j}{N_j}$$

$$R_j = \frac{Q_j}{N_j} - \frac{L_j L_j^t}{N_j^2}$$

$$W_j = \frac{N_j}{n}$$

where $N_j = |Xj|$ and we take diagonal of$L * L^T$ and Q, which can be treated as vectors instead of a matrix. K-means assumes spherical Gaussians (i.e. dimensions have the same variance). Centroids Cj are generally initialized with k random points. The algorithm iterates executing a couple of steps starting from some initial solution until cluster centroids become stable.

Step 1 determines the closest cluster for each point and adds the point to it. That is, this step determines cluster membership. We used Euclidean distance to determine the closest centroid to each point x_i which is defined as $d(x_i, C_j) = (x_i - C_j)^t(x_i - C_j)$

Step 2 updates all centroids Cj by averaging points belonging to the same cluster. The cluster weights Wj and diagonal covariance matrices Rj are also updated based on the new centroids. The quality of a clustering solution is measured by the average

quantization error $q(C)$, defined by Ordonez and Omiecinski [31] (also known as distortion and squared reconstruction error). The lower is the value of $q(C)$, the better the quality of clustering. $q(C) = \frac{1}{n} \sum_{i=1}^{n} d(x_i, C_j)$

where $x_i \in X_j$. This quantity measures the average squared distance from each point to the centroid of the cluster to which it belongs, according to the partition into k subsets. The K-means algorithm stops when centroids change by a marginal fraction in consecutive iterations which is measured by the quantization error. K-means is theoretically guaranteed to converge with decreasing $q(C)$ at each iteration, but it is customary to set a threshold on the number of iterations to avoid excessively long runs.

Chapter 5

Algorithms based on Summarization Matrices

5.1 Algorithm to Compute Non-Diagonal/Diagonal Gamma

This is the main algorithm which calls the two forms of Gamma matrix based on the type of model that we are computing. And also, based on the type of input dataset, that is, whether it is dense or sparse, it calls different type of Non-Diagonal matrix computation.

/* Input data is a .csv file in secondary storage*/
/* Output is any one of the Gammas */
User Input: $input = "filename.csv"$
System input: $chunk\_size = 75\%$(amount of RAM (main memory) allocated to R at that time)
Output: $final\_gamma$
$chunk\_object = chunkerFunction(input, chunk\_size)$
$chunk\_number = 1$
while *chunk_object* $\neq EOF$ **do**
  chunk = read next chunk
  partial_gamma_list[chunk_number] = **compute_Gamma***(chunk,model)*
  chunk_number++
  discard (remove from memory) the current chunk
  point the chunk_object at the starting position of the immediate chunk
end
$final\_gamma = \sum_{i=1}^{chunk\_number-1} partial\_gamma\_list[i]$

function **compute_Gamma***(chunk, model)*
  if $model \in (LR, PCA)$ **then**
    if *chunk is sparse* **then**
      Non-DiagonalGamma$_{Sparse}$*(chunk)*
    end
    if *chunk is not sparse* **then**
      Non-DiagonalGamma$_{Dense}$*(chunk)*
    end
  end
  if $model = NB$ **then**
    Diagonal_Gamma_List*(chunk) /* Algorithm 3 */*
  end
  if $model = KM$ **then**
    clusterAssignedChunk = **assignCluster**$(chunk)$*/* *Algorithm 2 */*
    Diagonal_Gamma_List*(clusterAssignedChunk) /* Algorithm 3 */*
  end
end function
/ function called back in main algorithm */*

Algorithm 1: Scalable Algorithm to compute Non-Diagonal/Diagonal Gamma.

5.2 Algorithm assignCluster

This algorithm takes a chunk of data, number of clusters (k) as given by the user and list of k cluster centroids as inputs. Then, it assigns the cluster id which is closest to a given point, in our case, a row of the chunk based on Euclidean Distance between the cluster centroids and that point. Finally, return the list of cluster id's assigned to the rows of the chunk.

/* Inputs are chunk, number of clusters and list of cluster$_c$*entroids* */
/* Output is List{*cluster id for each row of input chunk*} */

User Input: $k = number\_of\_clusters$
System Input: $X = chunk$
$cluster\_centroid\_list = list\{list\{cluster\_centroid_{attribute}\}_k\}$

Output: *cluster_id* vector

vector cluster_id(k)
for $i = 1 \ldots X.nrows$ **do**
  vector $sqre\_diff\_sum(k)$
  for $j = 1 \ldots k$ **do**
    $cluster\_centroid_j = cluster\_centroid\_list[j]$
    $sqre\_diff\_sum[j] = \sum(X[i,:] - cluster\_centroid_j)^2$
  end
  $cluster\_id[i] = which\_min(sqre\_diff\_sum)$
end
/* return the *cluster_id* list */

Algorithm 2: *assignCluster* Algorithm.

5.3 Diagonal Gamma List Algorithm

This algorithm is called from the main one to compute the list of Gammas one for each class/ cluster. This is utilized only for models like Naive Bayes and K-means which require one summarization matrix for a given class-label or a determined cluster, as mentioned in earlier sections.

/* Input data chunk contains the class label column as the last one */
Data: X = input data chunk
Output: $\{ \Gamma_1, \Gamma_2, \Gamma_3 \ldots \Gamma_k \}$
$d = X.ncols$
/* extract the class labels column and find unique values out of it */
$g = unique(X[, d])$ /* g represents the unique class_labels / cluster numbers
$gamma = list()$
$g_{index} = 1$ /* g_{index} represents the class_label number/ cluster number */
for $i \in g$ **do**
    /* get all the rows from X which match the class label i except the class labels column */
    $X_g = X[X[d] == i, -d]$
    if $X_g != NULL$ **then**
        $gamma[g_{index}] =$ **Diagonal_gamma**$(transpose(matrix(X_g))$
        /*Algorithm 4 */
        $g_{index} = g_{index} + 1$
    end
end
/* we return list of gammas back to the main algorithm */

Algorithm 3: *Diagonal_Gamma_List* Algorithm to return list of Gammas, each of which represent Gamma for one class_label/cluster.

5.4 Diagonal Gamma Algorithm

This is the algorithm where the actual computation of the summarization matrix happens. This algorithm is invoked multiple times by the Diagonal_Gamma_list algorithm in a single execution to compute a single summarization matrix irrespective of whether it is computing a classification or clustering model.

Data: X_g =subset of data with specific class label
Output: Γ_{diag} for one class_label/cluster
$N_g = X_g.cols$; $d_g = X_g.rows$;
for $i = 1 \ldots N_g$ **do**
  $x_i = list(d_g + 1)$
  $x_i[1] = 1$
  for $a = 1 \ldots (d_g)$ **do**
    $x_i[a + 1] = X_g[a, i]$
  end
  for $a = 1 \ldots d + 2$ **do**
    $\Gamma_g[a, 1] = \Gamma_g[a, 1] + x_i[a] * x_i[1]$
    if $a! = 1$ **then**
      $\Gamma_g[a, a] = \Gamma_g[a, a] + x_i[a] * x_i[a]$
    end
  end
end
for $a = 1 \ldots (d_g + 1)$ **do**
  for $b = (a + 1) \ldots (d_g + 1)$ **do**
    $\Gamma_g[a, b] = \Gamma_g[b, a]$
  end
end
/* return Γ_g with dimensions $(\mathrm{d}_g + 1) \times (d_g + 1)$ * /

Algorithm 4: *Diagonal_Gamma* Algorithm.

5.5 Time and Space Complexity Analysis

From [3], it is clear that the time complexity for the phase 1 of the Non-Diagonal Gamma with dense data as input is $O(d^2n)$ and sparse data as input is $O(k^2n)$, assuming k entries in x_i are non-zero on an average. In phase 2, we compute the machine learning models based on the Gamma from phase 1. So, time for phase 2 doesn't depend on n and is $\Omega(d^3)$, which for a dense matrix may approach $O(d^4)$, when the number of iterations in the factorization numerical method is proportional to d. This Non-Diagonal Gamma is used by models like Linear Regression and PCA.

A separate Gamma function, Diagonal Gamma, is used owing to the fact that a major set of the traditional Non-Diagonal Gamma has little-to-no utility for models like Naïve Bayes and K-Means. Time complexity of Diagonal Gamma computation is $O(dn)$ as we compute only L and diagonal of Q of the whole Non-Diagonal matrix. This time complexity applies for all the models utilizing the Diagonal Gamma except K-Means. Since K-Means requires the dataset to be read multiple times,let it be m, the time complexity would be $O(dnm)$. When we come to the space complexity, both Diagonal and Non-Diagonal Gamma occupy the same $O(d^2)$ space in the physical memory. In conclusion, we can say that Diagonal Gamma is faster than the Non-Diagonal Gamma.

Chapter 6

Experimental Evaluation

In this section, we present the experimental evaluation of our functions and the models built upon Diagonal and Non-Diagonal Gamma. In that sense, firstly we give a brief overview on the hardware and system configuration . Then, we give some information on the softwares that we used and explain what datasets we used for the experiments

Then, we move on to presenting the experimental results for each of the four models which utilize Non-Diagonal and Diagonal Gamma. In that respect, we try to prove that model computation using our package in R provides accurate results by comparing them with current best packages in R in Section 6.2. After that, we provide a brief explanation on what impact that our optimized package has on the computation of Machine Learning models in R in Section 6.3. Then, we try to prove that our package in R outruns the same built under a columnar database *Vertica*,

Table 6.1: Hardware and Operating System.

Item	**Local**
OS	Linux Ubuntu
CPU	4 cores:2.83GHz
Nodes	1
RAM	4 GB
Storage	294GiB

a language and environment for statistical analysis like *R* and distributed data processing engine like *Spark* in Section 6.4. Then, we show that the current routines in R language which compute the aforementioned models fail for large datasets, which is not the case with our scalable package.

6.1 Experimental Setup

6.1.1 Hardware

Here, we display the information about the hardware and the operating system configurations in the Table 6.1

6.1.2 Software

The softwares and packages that we used for our research are presented in this section briefly. Firstly, we used RStudio as an Integrated Development Tool (IDE) to develop this package. RStudio includes a console, syntax-highlighting editor that

supports direct code execution, as well as tools for plotting, history, debugging, and workspace management. R provides a simple but elegant package development environment which also allows us to create RCpp packages.

Secondly, we utilized the package *chunkR* which plays an important role in solving the main memory limitation in R. This package read tables chunk by chunk using a C++ backend and a simple R interface and allows a user to read long text or comma separated valued tables in chunks. First, we create a chunker object using the chunker function with the path to a file and others as arguments. Then, we can use the two functions defined in the package to manipulate the chunker object, namely, *next_chunk* and get_table to read the next chunk and retrieve the data respectively. There are other functions like *get_completed* and $get_colnames$ that are used to get the number of rows already read and the column names of the table respectively. Once it reads the entire file to completion, *next_chunk* returns false and an empty dataframe or an empty matrix is returned from the *get_table* function.

Thirdly, we used the RCpp package by R to seamlessly integrate C++ with R [32, 33] apart from providing both R functions and C++ classes for usage. This means that most of the R data types and created objects can be mapped to and from C++ equivalent types and objects and thus facilitating the users to be able to write new code with easier integration of third-party libraries. The data structures, types and objects include vectors, functions, environments and many more. Finally, we also used the package *iotools* which provide basic I/O tools for streaming and data parsing.

Table 6.2: Base datasets that are utilized and multiplied row-wise and column-wise.

dataset	**d**	**n**	**Description**	**Used for testing**
CreditCard	30	285k	increase in credit line? yes or no	Naïve Bayes
YearPredictionMSD	90	515k	there is rain or not	Linear regression and PCA
Iris	3	150	to know the flower species	K-Means

6.1.3 Datasets used

The datasets which are used for the experiments are listed below in Table 2. We also include the information about the algorithms which utilize these datasets. We used three different datasets to carryout the experiments. The first is the YearPredictionMSD dataset, second is CreditCard dataset and the last one is the Iris dataset, all of which are taken from the UCI Machine Learning repository. We replicated each of the datasets in order to get various combinations of n and d without altering statistical properties of the data. The first one was sampled and replicated to get combinations of *d=(9, 91)* and *n=(0.5M, 1M, 10M)*, second was replicated to get the combinations of *d=30* and *n= (0.2M, 1M, 10M, 100M)* and the third one is replicated to get *d=4*, *n=(0.1M, 1M, 10M, 100M)*.

6.2 Accuracy Validation

The Table 6.3 shows the results of the experiments that were performed using the two forms of Gamma. We compared the accuracy of model computations of our package with similar packages in R, which is a language and environment for statistical computing.

We implemented four models in our package, namely, Linear Regression, PCA, Naïve Bayes, and K-Means. For each model, we have a different way of measuring the accuracy with the common underlying metric being Relative Error.

For Linear Regression, we get an intercept and a Beta per attribute as an output for the model computed by Gamma functions. This is similar to the output given by *lm*, the preferred default routine in R for Linear Regression, for the same input dataset. We then compute the absolute differences among all the respective values of intercept and betas, from which we compute the relative differences. Finally, we report the maximum of the relative differences among the intercept and the betas in Table 6.3.

For PCA, we get a diagonal matrix, D, of Eigen values and two ortho-normal matrices, S and V, which are Eigen vectors of the given input matrix. Unlike other models, we do not compute PCA completely in Cpp as it gives inaccurate results. Rather, we use pure R routines to compute SVD of the correlation matrix generated from the Gamma functions. The values in D depict the relative importance of each column in S and V matrices. So, we imply on the point that, for the computation of

relative error, we take the values from D whose value is greater than 1. We first find the absolute differences among the pairs of corresponding values from the output of the Gamma functions and that of the default R routines, from which we compute the relative differences. We report the maximum of these relative differences in Table 6.3.

In Naïve Bayes, we build a model to predict the class labels for the test dataset. For that, we compute two separate Naïve Bayes models on the given input training dataset using the default R routine and the aforementioned Gamma functions. Consequently, we compute the prediction accuracy by finding the degree to which the predictions made by the functions of our package conforms to that from standard routine in R.

For K-Means, we group the input data into k clusters, where k is pre-defined by the user. We compute K-Means with both the default R routine and the previously discussed Gamma functions. The output from both the techniques have three vectors, namely, centers, radii and weights. We take the weight vectors, sorted in decreasing order, from both the models and obtain the respective absolute errors. We use this absolute error to compute the relative errors with respect to the weight vector of the model computed from the default R routine. We report the maximum value of relative error in the Table 6.3.

From Table 6.3, we understand that the results from the functions of our package are almost an exact match with the output given by the currently existing best packages in R.

Table 6.3: Accuracy of models on respective datasets.

Model	Maximum Relative Error	Dataset used
Linear regression	5.89E-10	YearPredictionMSD
PCA	4.75E-13	YearPredictionMSD
Naïve Bayes	0	CreditCard
K-Means	4.7E-2	Iris

6.3 Impact of Optimization

Table 6.8 and Table 6.9 summarizes the results of Principle Component analysis and Linear Regression on YearPrediction dataset. We can see that as the as the size of the dataset increases, the inbuilt R packages crash. But contrary to that, our package performs exceptionally well. One of the main reasons for the failure of inbuilt R packages can be attributed to the fact that it tries to load the whole dataset into memory, eventually resulting in untimely aborts of the program. But our package by passes this problem by not loading the entire dataset into the memory, instead breaking the dataset into chunks according to allocated memory which is further discussed in the below section.

From Table 6.10, even though the current packages in R scale well for small datasets, they result in untimely aborts for large datasets. As the size of dataset increases, the performance of our package improves greatly.

Table 6.6 gives the synopsis of Naïve Bayes model applied on the credit Card fraud dataset. Naïve Bayes is one of the basic machine learning models which is used initially by most statisticians and mathematicians due to its quick turnaround time.

Table 6.4: PCA experiment results on YearPrediction dataset from UCI Repository(Dense).

n	**d**	**R+** $\Gamma_{non-diag}$ **(dense)**	**R+** $\Gamma_{non-diag}$ **(sparse)**	**R**
0.5m	91	22	33	336
1m	91	66	80	575
10m	91	726	800	crashed
1m	9	9	9	21
10m	9	91	75	205
100m	9	1018	1020	crashed

Table 6.5: LR experiment results on YearPrediction Dataset from UCI Repository(Dense).

n	**d**	**R+** $\Gamma_{non-diag}$ **(dense)**	**R+** $\Gamma_{non-diag}$ **(Sparse)**	**R**
0.5m	91	22	36	276
1m	91	74	74	630
10m	91	720	828	crashed
1m	9	6	6	24
10m	9	91	69	285
100m	9	941	928	crashed

The results from our package are compared to that of the Naïve Bayes algorithm given by R. This also behaves in a similar manner showing that the current algorithm in R crashes for large values of n which is not the case with our package.

6.4 Performance Comparison

In order to better quantify the efficiency of our package, we built the aforementioned models in the spark environment and an equivalent Gamma implementation in Vertica too. While R failed to provide any sort of results for large datasets, spark and

Table 6.6: Naïve Bayes experiment results on Credit card fraud dataset from UCI Repository(Dense).

n	**d**	**R**+Γ_{diag}	**R**
0.2m	30	7	51
1m	30	40	158
10m	30	399	crashed
100m	30	1132	crashed

Table 6.7: K-Means experiment results on Iris dataset from UCI Repository(Dense).

n	**d**	**R** + Γ_{diag}	**R**
150	4	0	0
0.1m	4	6	0
1m	4	65	6
5m	4	380	crashed
10m	4	756	crashed

Table 6.8: PCA experiment results on YearPrediction dataset from UCI Repository(Dense).

n	**d**	**Vertica**	**R+ $\Gamma_{non-diag}$ (dense)**	**spark**
0.5m	91	46	22	67
1m	91	115	66	130
10m	91	1290	726	1074
1m	9	10	9	31
10m	9	110	91	286
100m	9	1560	1018	1780

Table 6.9: LR experiment results on YearPrediction Dataset from UCI Repository(Dense).

n	**d**	**R+ $\Gamma_{non-diag}$ (dense)**	**Vertica**	**spark**
0.5m	91	22	276	67
1m	91	74	115	130
10m	91	720	1290	1074
1m	9	6	10	31
10m	9	91	110	286
100m	9	941	1560	1780

Vertica, on the other hand, owing to its memory handling techniques is able to furnish tangible results albeit sub-par when compared to our package. The experiment results for the Naïve Bayes model in spark environment are not reported because spark did not have the flexibility to compute the model on negative values of attributes in our dataset. Linear Regression and PCA are the only models which are available in Vertica environment using Gamma matrix. So, we reported the results for those two models.

Table 6.10: K-Means experiment results on Iris dataset from UCI Repository(Dense).

n	**d**	$\mathbf{R} + \Gamma_{diag}$	**spark**
150	4	0	3.2
0.1m	4	6	7.5
1m	4	65	43.3
5m	4	380	1370
10m	4	756	3012

6.5 Discussion

Even though this model works efficiently for datasets with rows in the order of millions, it doesn't work as intended with the billion or higher rowed counterparts. This issue is magnified with the k-means algorithm as it requires multiple reads of the dataset before returning the final clusters. Notwithstanding, even with the long execution times, it still gives accurate results in contrast to the existing packages that result in untimely session aborts. As we see in the experimental results of k-means, the existing most efficient package for k-means model in R is aborted for a dataset with five million rows or higher. In a similar manner, even for Naïve Bayes, the most efficient package in R is aborted when a dataset with ten million rows is given as input while our solution returned accurate results within a reasonable amount of time.

Our solution adapts to the local machine and customizes the chunk size with respect to the available physical memory. The main drawback is that R cannot be easily parallelized unlike the Hadoop stack or other parallel systems to completely utilize the cores available in a system thus resulting in a decreased performance.

Chapter 7

Conclusions

We developed a package, Gamma, in R which is capable of computing four Machine Learning models, namely, Linear Regression, Principal Component Analysis, Naïve Bayes and K-Means with very high accuracy. This package eliminated the physical memory limitations of R by using efficient data chunking and summarization. Once we have the input dataset, we divide the data into chunks, whose size is derived dynamically based on the proportion of memory allocated by the processor to R. This benefits our package greatly by avoiding sudden aborts, as it utilizes only the part of memory allocated to R. For data summarization techniques, we adapted the previous work on Gamma matrix and implemented it in a serial R environment without using any external source of storage. In order to make R scalable and thereby improve the performance of computing models, we utilized the efficient C++ standards provided by RCpp package. We introduced specialized Diagonal Gamma

for models whose summarization could not be achieved with the traditional Non-Diagonal Gamma. The proposed Diagonal Gamma proved to be the best fit for models like Naïve Bayes and K-Means which require multiple summarization matrices to represent their classes/clusters. Then, we presented an algorithm that implements the construction of both Diagonal and Non-Diagonal Gamma. Provided the data and the machine learning model, the suitable gamma construction routine is called and model construction follows thereafter. We presented a reasonably good experimental section. Our package is more than two orders of magnitude faster than the current best packages in R for datasets with smaller number of attributes and is more than four orders of magnitude faster for datasets with an average number of attributes. As we go on increasing the value of n, number of rows in the dataset, the current packages in R fail when the input matrix does not fit in the physical memory (RAM). On the other hand, our package scales beyond the physical memory limits regardless of size of the dataset. We also compared our package with the equivalent ones taken from MLlib in spark. From the experiments, we can see that our package is two orders of magnitude faster than what MLlib delivered.

Even though our research proves that we can get better performance just with a single system for large datasets, there is enormous scope that we could expand it in multiple dimensions. We need to explore more Machine learning models on how they utilize this approach and generate the Summarization matrix. Also, we did not solve the problem of parallelizaton in R as it is not trivial with the existing packages like parallel. So, we can improve the performance of the package if we can achieve the parallelism across the cores of a single system. We can use Open MPI in R to achieve

more efficient parallelism from [34] rather than the existing packages like *parallel*. In contrast, similar languages like python, which is among the most popular ones for data analytics, can be parallelized easily. So, we look forward to implementing a similar package in python in expectation of getting a better performance than the one proposed. Furthermore, we can achieve even higher accuracy if we can combine multiple models on the dataset to arrive at a better model.

Bibliography

[1] S. García, S. Ramírez-Gallego, J. Luengo, J. M. Benítez, and F. Herrera, "Big data preprocessing: methods and prospects," *Big Data Analytics*, vol. 1, no. 1, p. 9, 2016.

[2] C. Ordonez, Y. Zhang, and W. Cabrera, "The Gamma operator for big data summarization on an array DBMS," *Journal of Machine Learning Research (JMLR): Workshop and Conference Proceedings (BigMine 2014)*, vol. 36, pp. 61–96, 2014.

[3] C. Ordonez, Y. Zhang, and W. Cabrera, "The Gamma matrix to summarize dense and sparse data sets for big data analytics," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 28, no. 7, pp. 1906–1918, 2016.

[4] Y. Zhang, C. Ordonez, and W. Cabrera, "Big data analytics integrating a parallel columnar DBMS and the R language," in *Proc. of IEEE CCGrid Conference*, 2016.

[5] M. A. Rehab and F. Boufares, "Scalable massively parallel learning of multiple linear regression algorithm with mapreduce," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 2, pp. 41–47, 2015.

[6] C. Anagnostopoulos and P. Triantafillou, "Efficient scalable accurate regression queries in in-dbms analytics," in *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pp. 559–570, 2017.

[7] G. V. d. Oliveira and M. C. Naldi, "Scalable fast evolutionary k-means clustering," in *2015 Brazilian Conference on Intelligent Systems (BRACIS)*, pp. 74–79, 2015.

[8] R. A. Horn and C. R. Johnson, *Matrix Analysis*. New York, NY, USA: Cambridge University Press, 2nd ed., 2012.

[9] R. Masashi Fukuda and T. Abro, "Linear, quadratic, and semidefinite programming massive mimo detectors: Reliability and complexity," *IEEE Access*, vol. 7, pp. 29506–29519, 2019.

[10] M. C. de Almeida, E. N. Asada, and A. V. Garcia, "On the use of gram matrix in observability analysis," *IEEE Transactions on Power Systems*, vol. 23, no. 1, pp. 249–251, 2008.

[11] Y. Lai, R. Orlandic, W. G. Yee, and S. Kulkarni, "Scalable clustering for large high-dimensional data based on data summarization," in *2007 IEEE Symposium on Computational Intelligence and Data Mining*, pp. 456–461, 2007.

[12] J. B. Kulkarni, A. A. Sawant, and V. S. Inamdar, "Database processing by linear regression on gpu using cuda," in *2011 International Conference on Signal Processing, Communication, Computing and Networking Technologies*, pp. 20–23, 2011.

[13] N. Senavirathne and V. Torra, "Approximating robust linear regression with an integral privacy guarantee," in *2018 16th Annual Conference on Privacy, Security and Trust (PST)*, pp. 1–10, 2018.

[14] R. Vilalta and I. Rish, "A decomposition of classes via clustering to explain and improve Naive Bayes," in *Proc. ECML Conference*, pp. 444–455, 2003.

[15] M. Balouchestani and S. Krishnan, "Advanced k-means clustering algorithm for large ecg data sets based on a collaboration of compressed sensing theory and k-svd approach," *Signal, Image and Video Processing*, vol. 10, no. 1, pp. 113–120, 2016.

[16] C. Elkan, "Using the triangle inequality to accelerate k-means," in *Machine Learning International Conference*, vol. 20, pp. 147–153, 2003.

[17] M. Muhr and M. Granitzer, "Automatic cluster number selection using a split and merge k-means approach," in *2009 20th International Workshop on Database and Expert Systems Application*, pp. 363–367, 2009.

[18] Q. Lin, C. Ceja, M. Hsu, Y. Mou, and M. Xu, "Face search in a big data system," *Electronic Imaging*, vol. 2016, no. 11, pp. 1–5, 2016.

[19] V. Raychev, M. Musuvathi, and T. Mytkowicz, "Parallelizing user-defined aggregations using symbolic execution," in *Proceedings of the 25th Symposium on Operating Systems Principles*, pp. 153–167, ACM, 2015.

[20] S. Pitchaimalai, C. Ordonez, and C. Garcia-Alvarado, "Comparing SQL and MapReduce to compute Naive Bayes in a single table scan," in *Proc. ACM CloudDB*, pp. 9–16, 2010.

[21] R. Zhang and Y. Wang, "An enhanced agglomerative fuzzy k-means clustering method with mapreduce implementation on hadoop platform," in *2014 IEEE International Conference on Progress in Informatics and Computing*, pp. 509–513, 2014.

[22] A. Boukhdhir, O. Lachiheb, and M. S. Gouider, "An improved mapreduce design of kmeans for clustering very large datasets," in *2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA)*, pp. 1–6, 2015.

[23] L. Zhou, Z. Yu, J. Lin, S. Zhu, W. Shi, H. Zhou, K. Song, and X. Zeng, "Acceleration of naive-bayes algorithm on multicore processor for massive text classification," in *2014 International Symposium on Integrated Circuits (ISIC)*, pp. 344–347, 2014.

[24] C.-H. Lee, "A gradient approach for value weighted classification learning in naive bayes," *Knowledge-Based Systems*, vol. 85, pp. 71 – 79, 2015.

[25] M. Wu, D. Dai, Y. Shi, H. Yan, and X. Zhang, "Biomarker identification and cancer classification based on microarray data using laplace naive bayes model with mean shrinkage," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 9, no. 6, pp. 1649–1662, 2012.

[26] C. Keerthisinghe, H. Sun, Y. Takaguchi, D. Nikovski, and H. Hashimoto, "Machine learning based state-space approximate dynamic programming approach for energy and reserve management of power plants," in *2018 IEEE Innovative Smart Grid Technologies - Asia (ISGT Asia)*, pp. 669–674, 2018.

[27] J. Kacprzyk and G. Szkatula, "An integer programming approach to inductive learning using genetic algorithm," in *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, vol. 1, pp. 181–186 vol.1, 2002.

[28] C. Ordonez, W. Cabrera, and A. Gurram, "Comparing columnar, row and array dbmss to process recursive queries on graphs," *Information Systems*, 2016.

[29] J. Marin and C. Robert, *Bayesian Core: A Practical Approach to Computational Bayesian Statistics*. Springer, 2007.

[30] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. New York: Springer, 1st ed., 2001.

[31] C. Ordonez and E. Omiecinski, "Efficient disk-based K-means clustering for relational databases," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 16, no. 8, pp. 909–921, 2004.

[32] D. Eddelbuettel and J. J. Balamuta, "Extending r with c++: A brief introduction to rcpp," *The American Statistician*, vol. 72, no. 1, pp. 28–36, 2018.

[33] D. Eddelbuettel and R. Francois, "Rcpp: Seamless r and c++ integration," *Journal of Statistical Software, Articles*, vol. 40, no. 8, pp. 1–18, 2011.

[34] M. Schmidberger, M. Morgan, D. Eddelbuettel, H. Yu, L. Tierney, and U. Mansmann, "State-of-the-art in parallel computing with r," 2009.